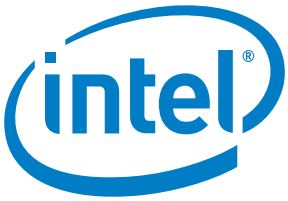


Sycamore Island (SCI)

Firmware/Software Architecture Specification

Version: 1.1 Status: Approved
Date: Jan 22, 2021
Document author : Malhotra, Rahil
Document List of Approvers :-
Functional Safety Architect : Malhotra, Rahil
Functional Safety Manager : Pingitore, Francesco

Applicable Standard: IEC 61508
Configuration : GC-EHL Development (Global Configuration)

Intel Top Secret

Contents

Figures

Tables

Revision History

Date	Revision	Author	Description
Jan-21	1.1	Malhotra, Rahil	- Changes for RTC: 310693, 319151, 319153 and 310515 - Fixed error handling flow diagrams - Addressed all known gaps and opens - Addressed feedback from SW FMEA
Nov-20	1.09	Malhotra, Rahil	Added MDS trace to SSA in External System ID/Name
Oct-20	1.08	Malhotra, Rahil	Updated 850181(ssa) to general text, reference RTC_340019
Sep-20	1.07	Malhotra, Rahil	Update with reference to traceability review fixes
Aug-20	1.06	Malhotra, Rahil	- Updated to "General Text": 590324, 590332, 863474, 590365, 590396, 590547 - Update of the trace review fixes
Aug-20	1.05	Malhotra, Rahil	Changes incorporated for Traceability to the SWSRs
Mar-19	1	Banginwar, Rajesh	All review feedback included. Production ready 1.0 release
Jan-19	1.0RC	Banginwar, Rajesh	Changes to many sections to accommodate ODCC updates, reorganization of FSTs and feedback from engineering as part of ongoing design and implementation phase.
Nov-18	0.82	Banginwar, Rajesh	Failure scenarios for ODCC explicitly discussed in section 4.3; DTI/2 timer discussion added in ODCC flows in section 4.3
Nov-18	0.81	Banginwar, Rajesh	Updated ODCC flows to address the PST change to 1msec. The flows now handle PST of any value of PST from 1msec on-wards. Majorit of changes are: Section 4.2 and 4.3 – all FSTs including ODCC; also reorganized the section. Section 5.1 – all Host SW stack level details.
Jul-18	0.8	Banginwar, Rajesh	All review comments addressed. 0.8 forma
Jun-18	0.8RC	Banginwar, Rajesh	Release candidate for 0.8 version. Out for review.
Feb-18	0.5	Banginwar, Rajesh	Post formal review – version 0.5 release.
Jan-18	0.5RC	Banginwar, Rajesh	Release candidate for 0.5 version
Nov-17	0.2	Banginwar, Rajesh	Initial release

1.0 Introduction

1.1 Purpose

This document describes the Firmware and Software level flows at system level from Sycamore Island (referred as SCI) point of view. This document defines design and architecture for SCI meeting all functional, security and FuSa requirements.

SCI is a soft IP that will get integrated in Elkhart Lake PCH – Mule Creek Canyon as first intercept. The IP is expected to be general purpose with future PCH platforms as intercept targets. The IP is being designed to help achieve the required SIL and ASIL levels for the Intel IOTG and ADG platforms. The first intercept Elkhart Lake (EHL) is targeted to achieve SIL-2 with the help of SCI. The safety integrity level of SCI will comply with SIL-2 (hardware random) and SC-3 (systematic) according to IEC 61508 standard.

This document will describe the IP as it plays a role in the following platform FuSa activities:

- Error monitoring
- Functional safety test orchestration
- And all other use cases targeted for the associated platform

The IP design will allow configuration and extensions. The current version of the IP will not allow OEM programmability, but various Intel platform teams may add newer FW modules as needed.

There is a good amount of reuse between two generations of Safety Islands – Sentinel Island and Sycamore Island. However to keep the document self-sufficient, this SAS will include all the reuse sections from the SNI SAS too.

1.2 Audience

The audience for this document is the FuSa Global Domain ISI WG, the SCI teams and related involved people in IOTG and SoC teams. Primary audience will be the development and validation engineering teams working on SCI.

1.3 Related Publications

Refer to documents in below table for additional and background information.

Table 1 Related publications

Document	Document Number / Location
ISO 26262-2018 (2nd edition DIS)	FuSa Global Domain Sharepoint
IEC 61508	FuSa Global Domain Sharepoint
Elkhart Lake Safety Concept	FuSa Global Domain Sharepoint
Elkhart Lake PAS	FuSa Global Domain Sharepoint
SCI HAS	HAS sharepoint
SCI PRD	DNG

Sentinel Island SAS	FuSa Global Domain Sharepoint
Elkhartlake STL Software Architecture and Design Spec	FuSa Global Domain Sharepoint
SCI SWSRs	FuSa Global Domain Sharepoint
Residual error calculation document	Residual Error

1.4 Terminology

A summary of terminology used in this document is outlined in the following table.

Table 2 Terminology

Term	Description
DCLS	(HW) Dual Core Lock Step
LCLS	(SW) Loosely Coupled Lock Step
ODCC	(SW) On Demand Cross Comparison
ISI	Intel Safety Island
DTI	Diagnostic Test Interval
CCF	Common Cause Failures
MBU	Multiple Bit Upset
TSN	Time Sensitive Network
fRSC	faultRobust Smart Comp
HRF	Hardware Random Failures
CoP	Cove Peak SoC
Atom	Intel CPU SoC family
PDC	Pedro Canyon PCH
EHL	Elkhart Lake platform
DO	Desert Oaks Platform
DNV-AD	Denverton-AD SoC
PUnit	Power controller unit of Atom
MCC	MuleCreek Canyon chipset used in EHL
PCODE	Firmware executing on PCU
DSW	Deep Sleep Well

Term	Description
uCode	Atom's cores microcode
B2P	BIOS to PCODE mailbox interface
ME	Manageability Engine
IE	Innovation Engine
ISI PUS	ISI Public Space
ISI PVS	ISI Private Space
1oo2D	One out of Two with Diagnostics
DC	Diagnostic Coverage
frSmartComp	Fault Robust Smart Comparator
FMM	Fault Management Module
DTF	Debug Trace Fabric
RAVDM	Register Access over VDM
VDM	Vendor Defined Message
HVM	High Volume Manufacturing
MCU	Microcontroller Unit
IPC	Inter Process Communication
PMC	Power Management Controller
LFM	Latent Fault Monitoring
JSL	Jasper Lake CPU
ARM CM3	ARM Cortex M3
fRCPU	Fault Robust CPU for Cortex M3
RBE	ROM Boot Extension
MEU	Manifest Extension Utility
IOSF-SB	IOSF SideBand
MEMWR	Memory Write
FSP	Field Support Package
FST	Functional Safety Test
FSTM	Functional Safety Test Manager

Term	Description
START DTI	Message Sent by ISI to Host at the beginning of every configured DTI interval
CONFIGURED_TEMP_ALERT_VALUE	Default: 110C
THERMTRIP_VALUE	Default: 135C
T_{CYCLE}	Periodicity at which the Customer's Safety Workload runs.
T_{MAX_STL}	Maximum Time period within which SCI must get consecutive STL results from Customer's safety workload. Refer Section 4 for more details
T_{MIN_STL}	Minimum Time period that must elapse before SCI get consecutive STL results from Customer's safety workload. Refer Section 4 for more details
T_{MAX_ODCC}	Maximum Time period within which SCI must get consecutive ODCC snapshots from Customer's safety workload. Refer Section 4 for more details
T_{MIN_ODCC}	Minimum Time period that must elapse before SCI get consecutive snapshots from Customer's safety workload. Refer Section 4 for more details
TO_{IDEAL_STL}	Ideal time difference between Customer workload sending 'Start Workload' and 'First STL results'. Refer Section 4 for more details
TO_{MAX_STL}	Maximum time difference between Customer workload sending 'Start Workload' and 'First STL results'. Refer Section 4 for more details
TO_{IDEAL_ODCC}	Ideal time difference between Customer workload sending 'Start Workload' and 'First ODCC snapshot'. Refer Section 4 for more details
TO_{MAX_STL}	Maximum time difference between Customer workload sending 'Start Workload' and 'First ODCC Snapshot'. Refer Section 4 for more details
T_{ISI_MAX}	The Maximum time ISI takes to process a command, and assert NOK if it detects an error.
T_{RES_MAX}	The Maximum time it takes for the Mailbox Physical channel to transfer a payload from Host to SCI Mailbox
DTB	Diagnostic-time budget : It refers to the fraction of PST allocated by the system developer. In case of HFT = 0, PST can be configured from 1ms up to the SCI watchdog configuration limit (21s). In case of HFT > 0, the PST can be set to a maximum time that the SCI watchdog configuration supports currently it is 21s.

Term	Description
STL_ALLOCATED_TIME	The time between execution of the largest Test Module and placing the results on the PCIeMB
PST	Used interchangeably with EHL Reaction time budget.

1.5 Known Gaps/Opens

Table 3 Open & Follow up Needs

Open & Follow up Needs	Notes

2.0 Guide to the Selection of Techniques and Measures

2.1 Required Techniques for Software design and development – software architecture design

From the IEC 61508-3:2010 Table A.2, and Annex C.1 for software systematic capability, the following parameters were derived and a mapping of its use in SCI firmware has been documented.

The rigor R factor R2 as suggested in IEC 61508-2 section C.1.2 has been followed.

Table 4 Reference Documents Techniques Employed for SIL 3

Technique	SIL3	Use by SCI FW
Fault detection	HR	Used
Error detecting and correcting codes	R	Used
Failure assertion programming	R	Used
Diverse monitor techniques (with separation between the monitor computer and the monitored computer)	R	Used
Diverse redundancy, implementing the same software safety requirements specification	---	Not Used
Functionally diverse redundancy, implementing different software safety requirements specification	R	Not Used
Stateless software design (or limited state design)	R	Not Used
Graceful degradation	HR	Not Used
Modular approach	HR	Used
Use of trusted / verified software elements (if available)	HR	Used
Forward traceability between the software safety requirements specification and software architecture	HR	Used
Backward traceability between the software safety requirements specification and software architecture	HR	Used

Technique	SIL3	Use by SCI FW
Structured diagrammatic methods	HR	Not Used
Semi-formal methods	HR	Used
Formal design and refinement methods	R	Not Used
Automatic software generation	R	Not Used
Computer-aided specification and design tools	HR	Used
Cyclic behavior, with guaranteed maximum cycle time	HR	Used
Time-triggered architecture	HR	Not Used
Static resource allocation	HR	Used
Static synchronization of access to shared resources	R	Used

The following subsections gives the justification for using or not using the required Techniques for SIL 3.

2.1.1 Fault Detection

The SCI FW is designed to detect errors in different platform components e.g. faults in the CPU core where safety application will run, data channel where safety data flows. IEC clause C.3.1 pertains to fault detection of the software code. As the FW is responsible for detection of all such errors in the platform, it also ensures that if there is a software fault in its own code, that fault also gets detected. For that, it uses techniques like input/output data validation so that no firmware component uses invalid data. Firmware has temporal and program flow monitoring which ensures that each FW component is running within its time budget and following its statically defined flow.

2.1.2 Error Detecting and Correcting Codes

The SCI FW implements error detecting codes in the form of CRC. It detects data corruption for data received from other FW components in SOC over the channel. Communication with most of the peripherals is E2E protected. Corruption of memory is prevented/detected by ECC.

2.1.3 Failure Assertion Programming

This technique is employed. Assertion checks are made, failure to fall in expected range causes error to be reported from various firmware components which would eventually cause system to go to safe state. E.g. in switch statement, for all expected cases, required handling is there. Default case is treated like an error.

2.1.4 Diverse Monitor Techniques

Diverse monitoring technique is implemented where the host acts as a watchdog for the SCI. In case the SCI is unable to perform its safety functions, the host will be able to trigger a platform reset there by taking the system to a safe state.

2.1.5 Stateless Software Design (or Limited State Design)

The architecture of SCI FW implements a state less design, where every PST, a set of diagnostic routines are executed and action taken based on error reported.

2.1.6 Graceful Degradation

The SCI FW does not prioritize the functions available. In case of a failure of any one of the safety mechanism, the system enters a safe state and will remain unavailable. Using 1oo2D configuration increases the availability by providing a redundant channel to function.

2.1.7 Modular Approach

This technique is being used by the SCI FW. The interfaces between each module is well defined. The coding guidelines provides rules for limiting the number of lines in a method and in for a file. The module interfaces are clearly defined in SAS/MDS documents.

2.1.8 Use of Trusted / Verified Software Elements (If Available)

The SCI FW uses third party RTOS (threadX). It is functional safety compliant software implemented according to ISO/IEC standards and certified by TUV SUD to be developed as per ISO/IEC standards.

2.1.9 Forward Traceability

Forward traceability from the SW safety requirements and the software architecture are captured in the Jazz DNG tool.

2.1.10 Backward Traceability

Backward traceability from the software architecture and the SW safety requirements are captured in the Jazz DNG tool.

2.1.11 Structured Diagrammatic Methods

Structured diagrammatic methods are not being used since it is more formal than necessary and adds little value due to the nature of the SCI FW. Semi-formal methods such as flow diagrams are in use within this document to describe these control/data flow mechanisms as necessary.

2.1.12 Semi-formal Methods

The Design of the SCI FW uses Semi-formal methods in various places in this document in the form of flow diagrams to help eliminate any ambiguity.

2.1.13 Formal Design and Refinement Methods

This technique does not apply to the SCI FW. Semi-formal methods such as flow diagrams are in use within this document to describe these control/data flow mechanisms as necessary.

2.1.14 Automatic Software Generation

The SCI FW does not employ this technique.

2.1.15 Computer-aided Specification and Design Tools

For this technique, we are using the Microsoft Office Professional, Microsoft Visio Professional, and Jazz toolchain.

2.1.16 Cyclic behavior, with guaranteed maximum cycle time

Cyclic and event driven methods are used in the SCI FW implementation. Some FSTs completes their execution within DTI window. There is a thread that acts as supervisor that monitors correct and in time functioning of all FST threads. If that does not happen, this thread takes the system to safe state. For some other FSTs which have their own timing requirements are not monitored for their timing by supervisory thread. In those cases, supervisory thread just monitors FSts are alive and responding. If for some reason, supervisory thread itself hangs, watchdog does not reset which causes system to go in safe state.

2.1.17 Time-Triggered Architecture

This technique is not used by the SCI FW. The RTOS schedules each ready thread periodically. If a thread is not ready within DTI window and not completes its assigned task within that, system goes to safe state.

2.1.18 Static Resource Allocation

Byte pools for various modules are created at the startup. All modules allocate/use memory from within that byte pool at the startup. In specific cases, memory is dynamically allocated from within that byte pool. In case of going beyond the limits of byte pool, NOK is generated.

2.1.19 Static Synchronization of Access to Shared Resources

Access to shared resources is synchronized in run time using RTOS synchronization primitives.

3.0 Firmware Architecture

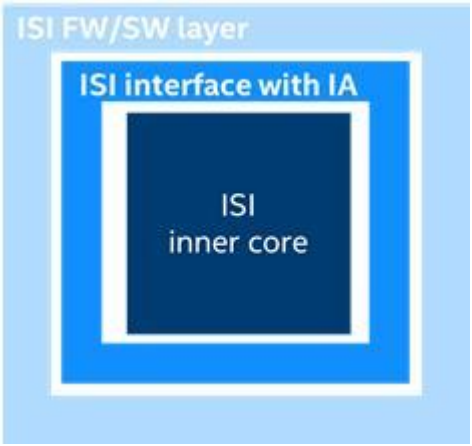
This section will go into all the details associated with each of the function/component mentioned in the earlier section. Since the document is primarily focused on Sycamore Island, we will cover only the components that are interacting with Sycamore Island. In the current revision of the document, the section will cover only the Elkhart Lake platform.

3.1 The ISI and its overall HW/FW architecture

The Intel Safety Island ("ISI") can be represented in three different layers:

- ***ISI inner core***: the HW core centered on an HW lock-step processor; ISI inner core is the central microcontroller unit in charge of monitoring and error management and needs to be designed in order to satisfy the assumed uses cases.
- ***ISI outer-core/interface with IA***: the connections with Intel Architecture, e.g. how to interface ISI with Machine Check Architecture and other error signals, how to provide the physical path for core2core comparison (e.g. ODCC), how to interface with HVM or other protected resources in IA, how to interface with the MCU or other platform components the ISI has to communicate for safety operations.
- ***ISI FW/SW layer***: the FW/SW pieces to implement the functions required to execute the ISI functions.

Figure 1 ISI overall architecture



3.2 Firmware Architecture Overview

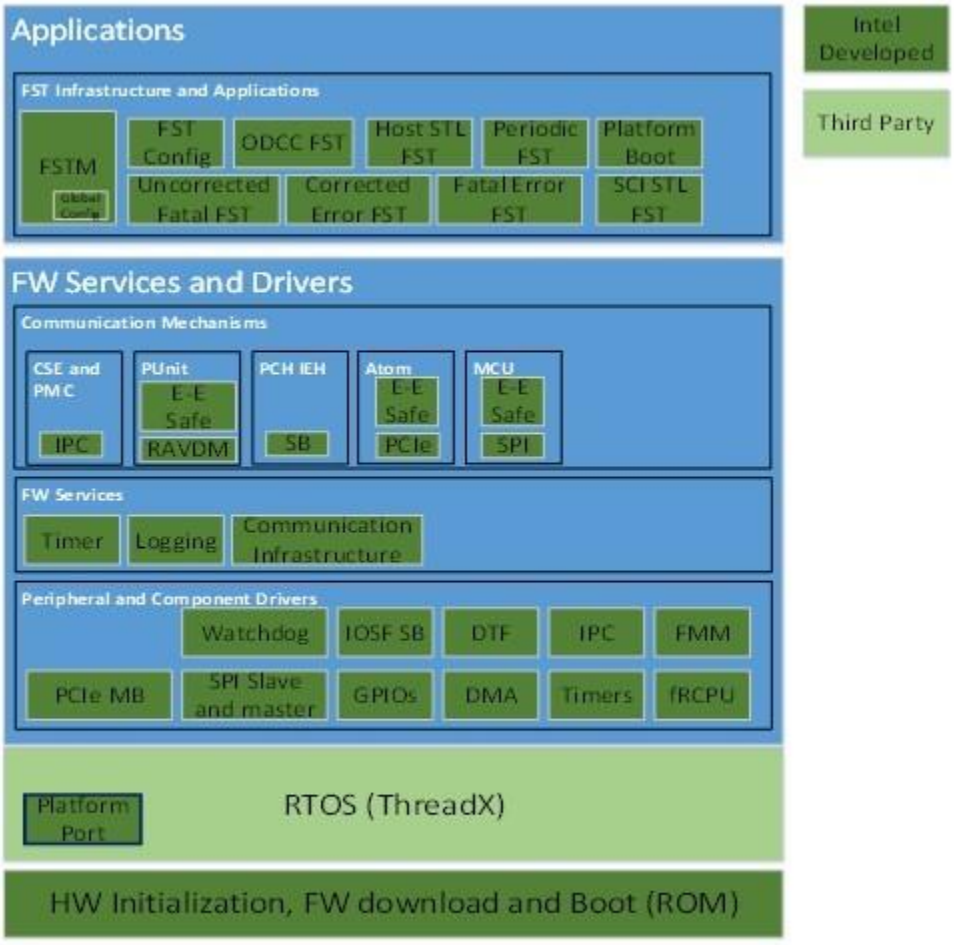
The ISI firmware at a high level has following components:

- Initial boot and HW initialization
- RTOS – ThreadX with allowed changes for porting to the platform
- Various FW services. Let's categorize them in 3 different buckets
    - Various platform drivers needed to communicate with various interfaces

- FW services like diagnostic data logging, timer etc
- Communication infrastructure for various components like safety MCU, Atom, PCH etc.

- Various components driving the test orchestration called as FSTs (Functional Safety Tests).
  - FSTM = Functional Safety test manager
  - FST = Functional safety test
- Following stack diagram depict various components mentioned above:

Figure 2 SCI FW Block Diagram



3.3 Firmware Infrastructure and Services

3.3.1 Timer Service

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 590275	Accepted		SIL 3	In SCI, the Proc SS implements a new Timer block which provides following three types of Hardware timers: 1. RTOS Timer 2. Periodic timer 3. GP (general purpose) Timers The RTOS timer is a free running timer which gives registers to read the time (in nsecs) since last boot. ARM Cortex M3 has a Systick timer that can generate periodic timer interrupts. This is used for the RTOS Scheduler for time slicing. It uses the same clock source as the CPU, running at 200 MHz. The requirement to support 16 WCET/BCET requests from the host is fulfilled by the GP Timers. But since the HW GP Timer instances are limited by the number of available IRQ lines in the HW (12 HW GP Timers), the FW needs to provide a virtualized Timer service (NUM_MAX_TIMER_REQUIRED=60) to fulfill the WCET/BCET requirements. Also, we plan to use the same virtualized timer service in all the FW components – such as FSTs for PST measurement, in interfaces for timeout calculation etc. Hence based on above requirements and limitations, we propose the following approach to virtualize the timer service: 1. NUM_OF_TIMER_PER_GP_TIMER_HW = NUM_TIMER_INSTANCE / MAX_GP_TIMER_HW 2. For each incoming timer request the Timer HAL will allocate a virtual timer from the next free HW GP Timer till NUM_OF_TIMER_PER_GP_TIMER_HW virtual timers are bound to that HW GP Timer. 3. From the virtual timers running, bound on a particular HW GP Timer, the timer with the lowest expiry time will be actually bound to the GP Timer instance. 4. For other virtual timers that are not bound to the HW, time accounting (elapsed time and expiry time) would be done using the RTOS Timer. Below flow diagrams capture these flows in detail. Please refer them:		N/A	Test	Intel Confidential

Figure 3 Create Timer

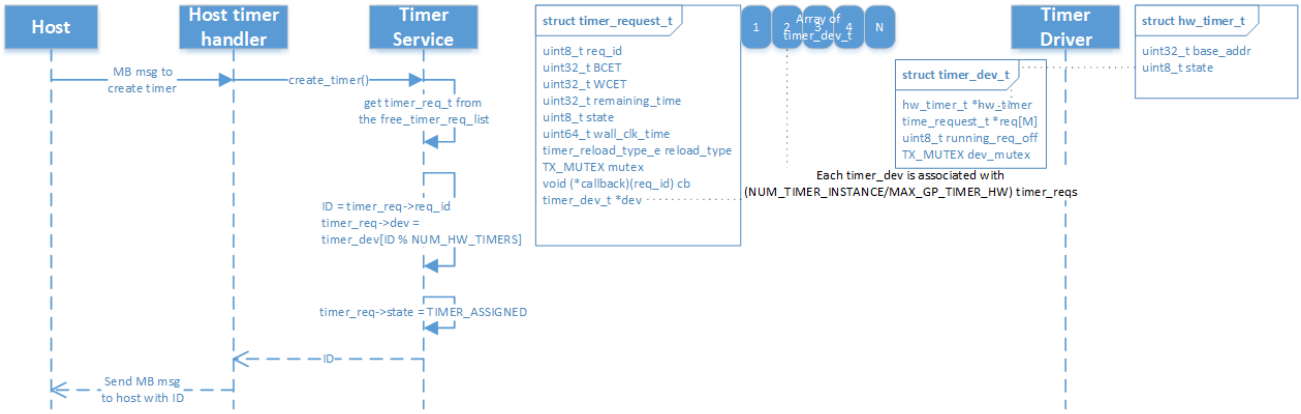


Figure 4 Configure timer

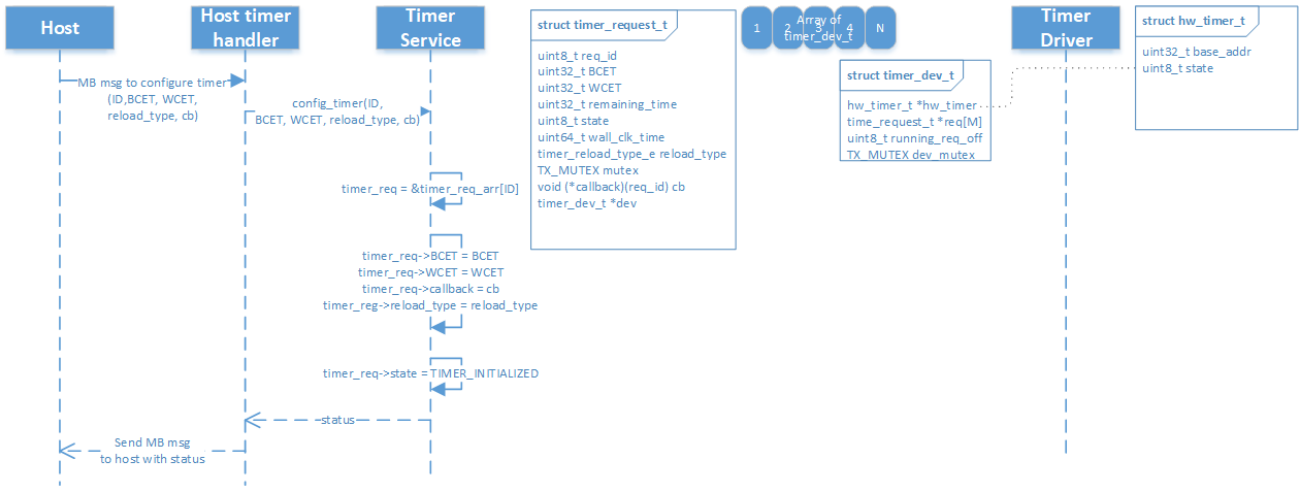


Figure 5 Start timer

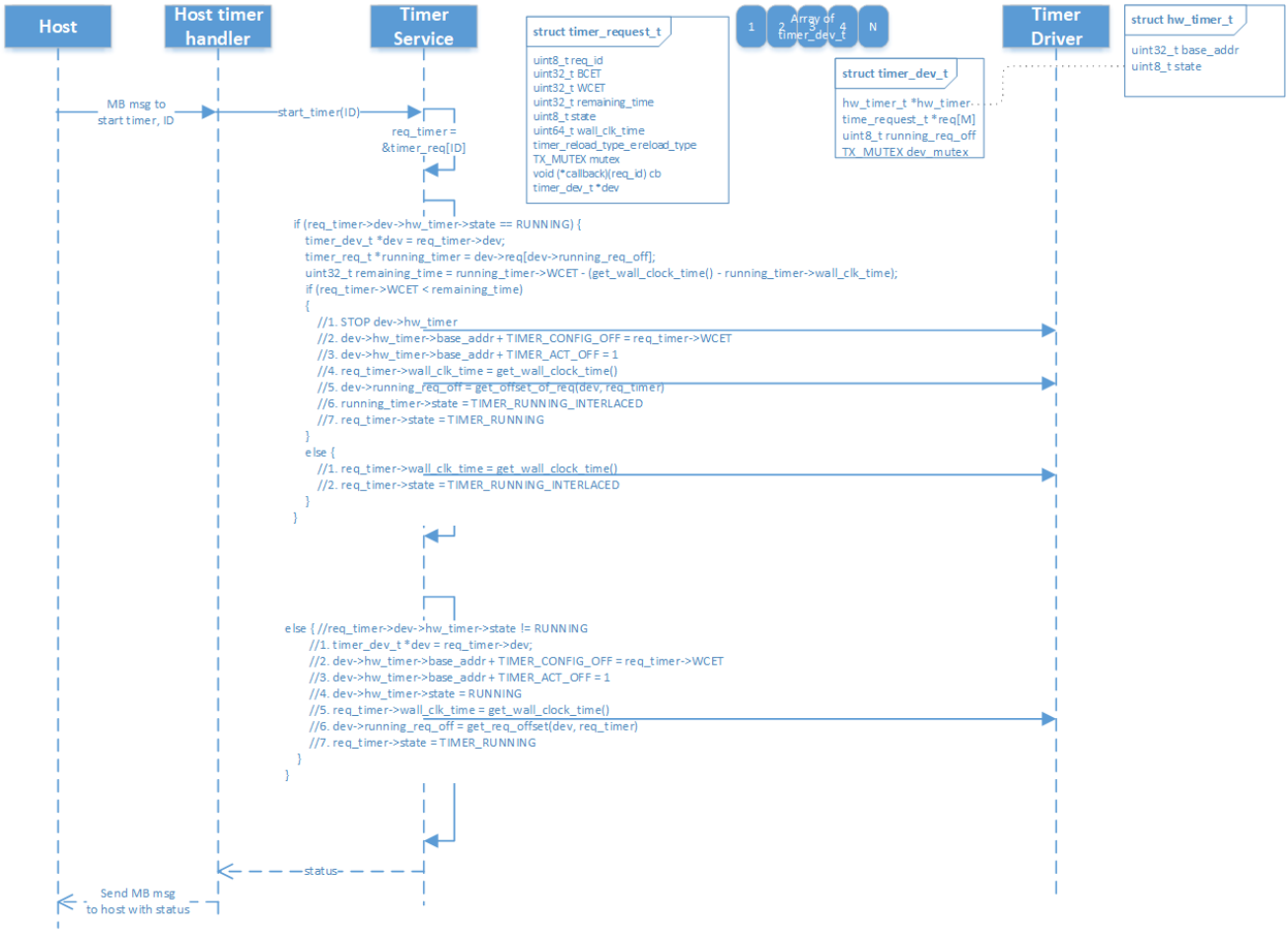


Figure 6 Stop Timer

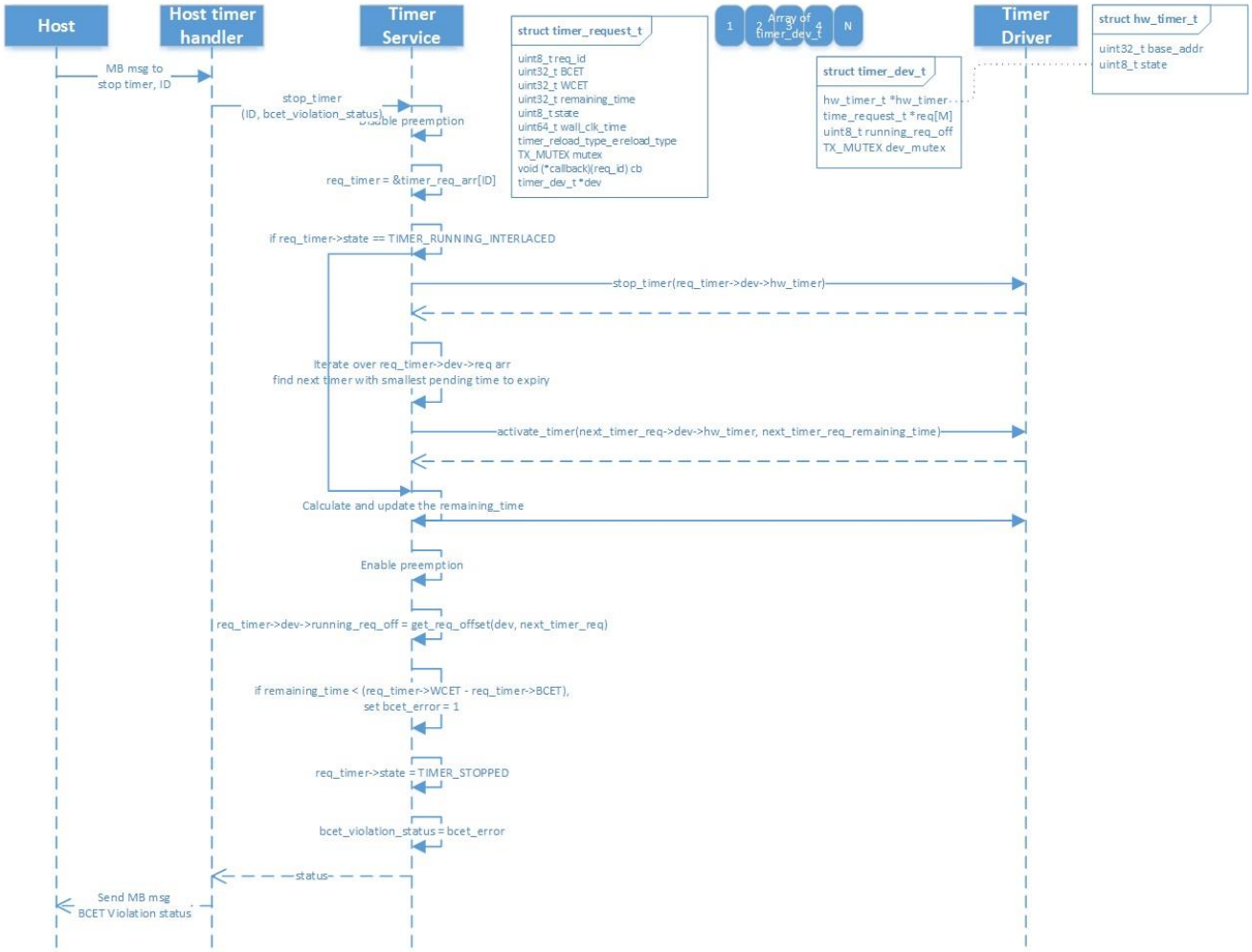


Figure 7 Timer expiry

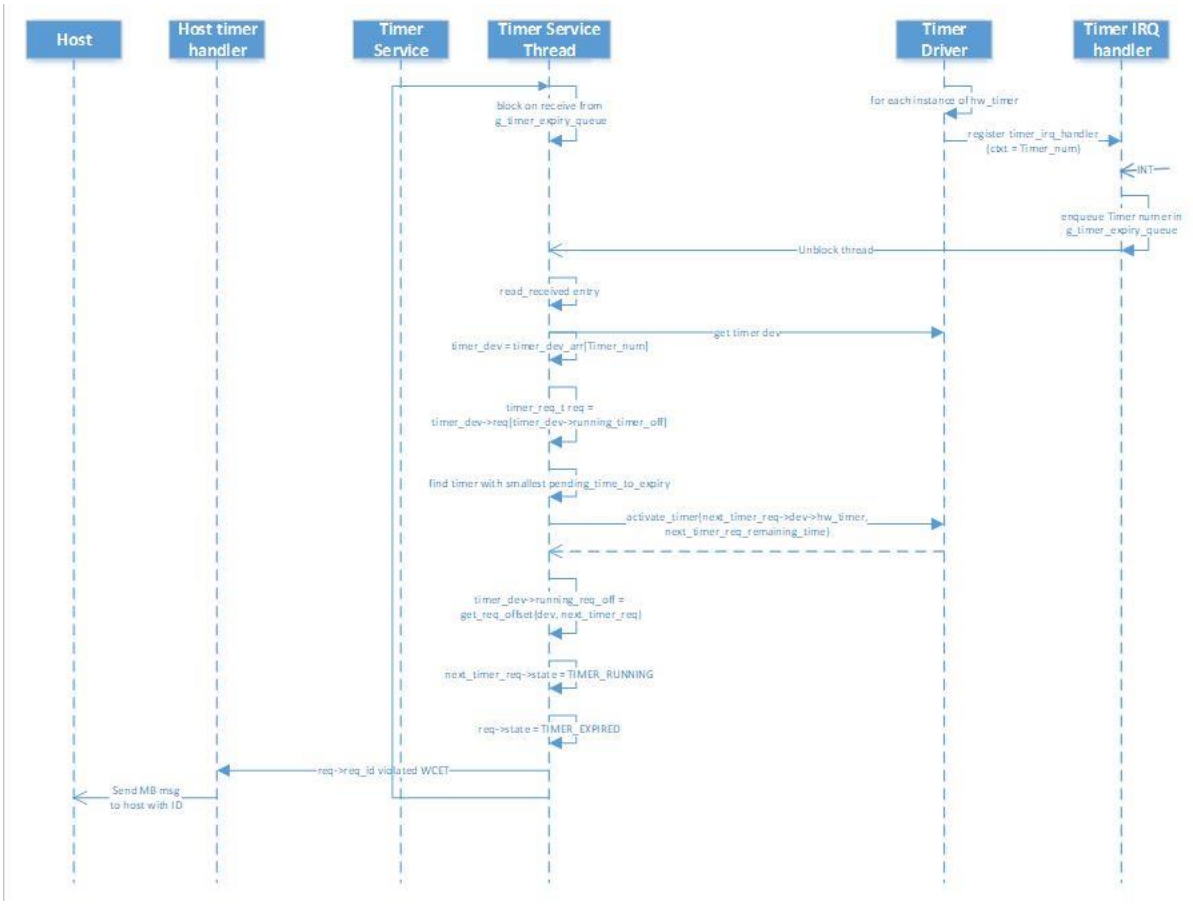


Figure 8 Delete Timer

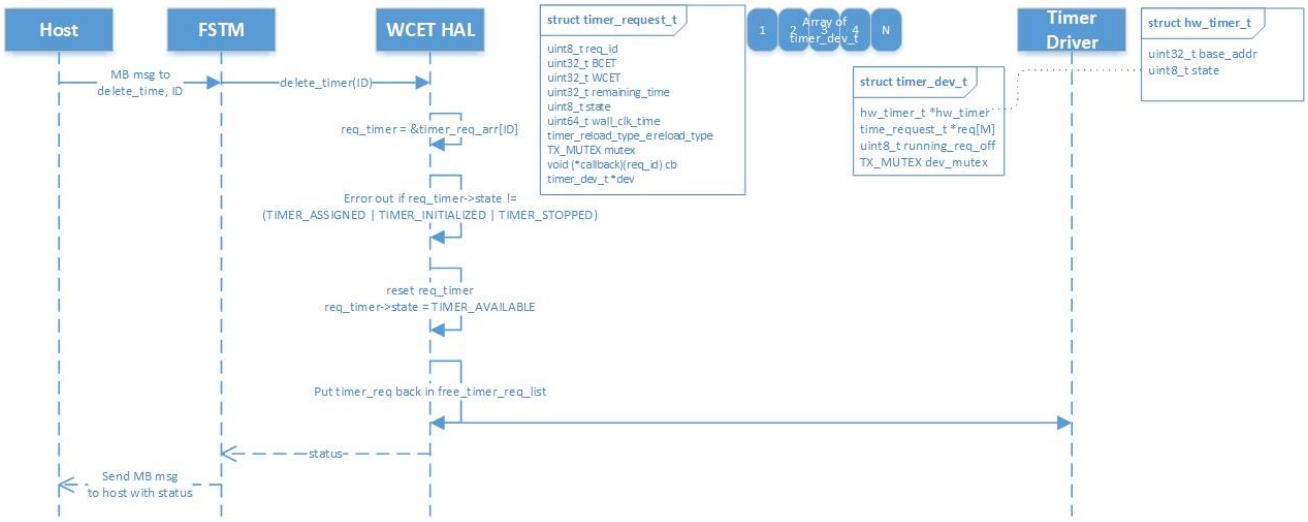


Figure 9 Pause Timer

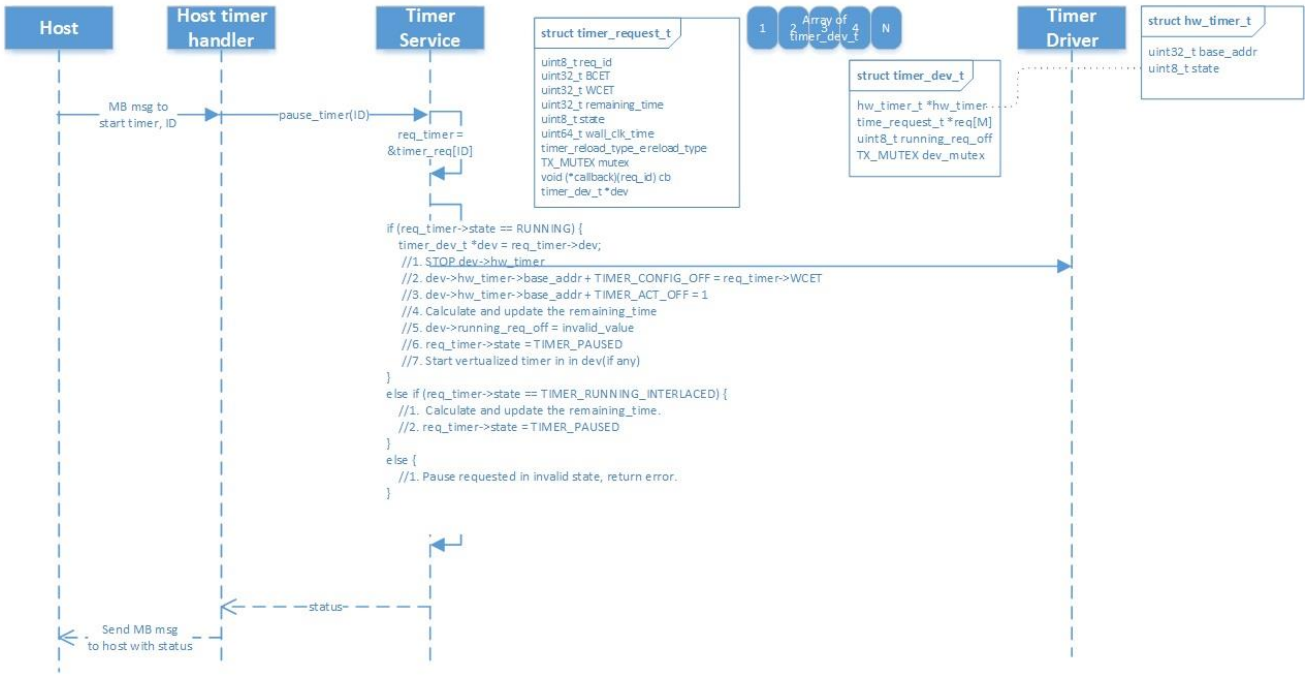
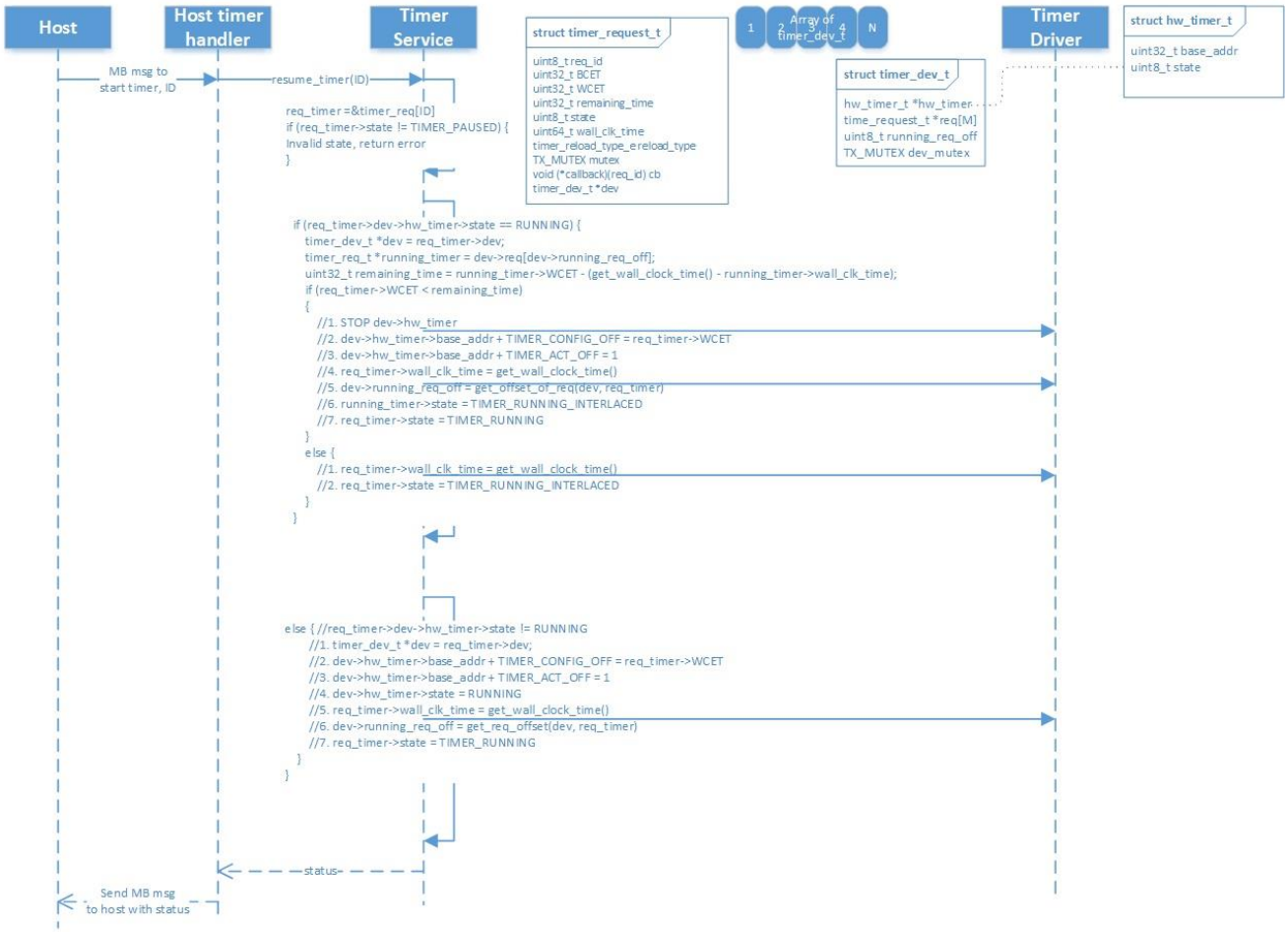


Figure 10 Resume Timer



The following set of flow diagrams show how the Timer service for the FSTs is built on top of the WCET Timer HAL. Each FST is expected to use these services for any timer requirements instead of using the ThreadX Timer service. Through these APIs the FST will be able to register a call back function with the WCET HAL, which will get invoked once the timeout value associated with the timer expires.

Figure 11 Create FST Timer

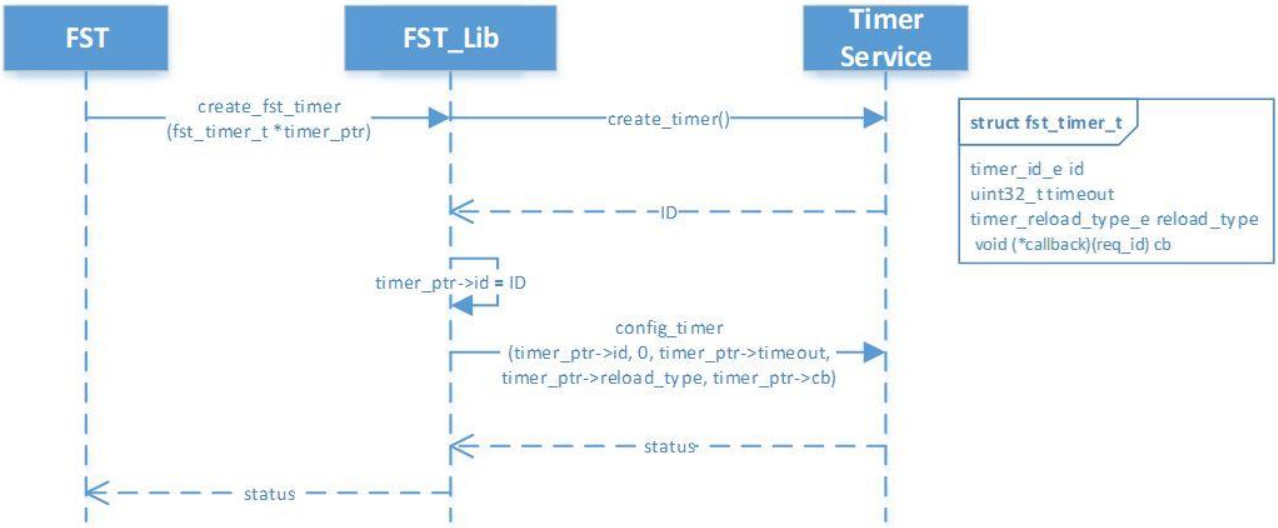


Figure 12 Activate FST timer

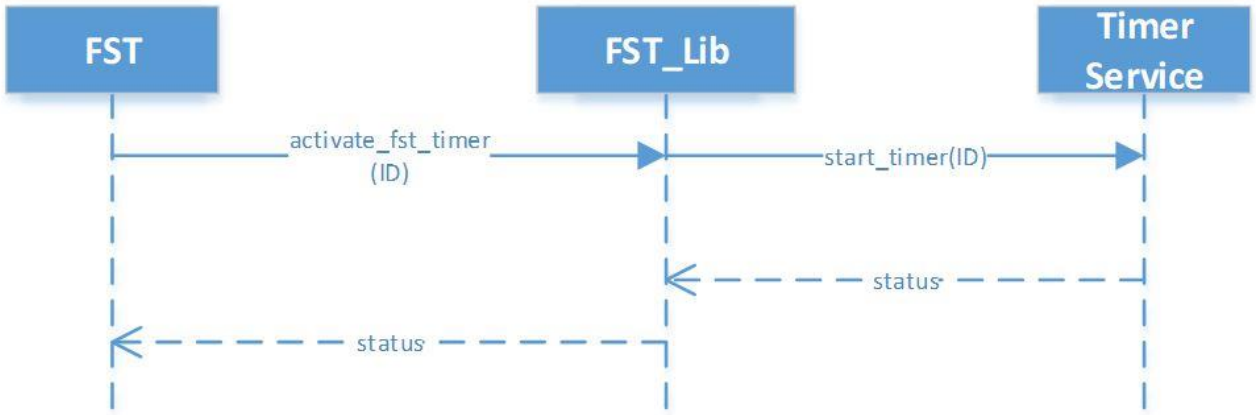


Figure 13 Deactivate FST timer

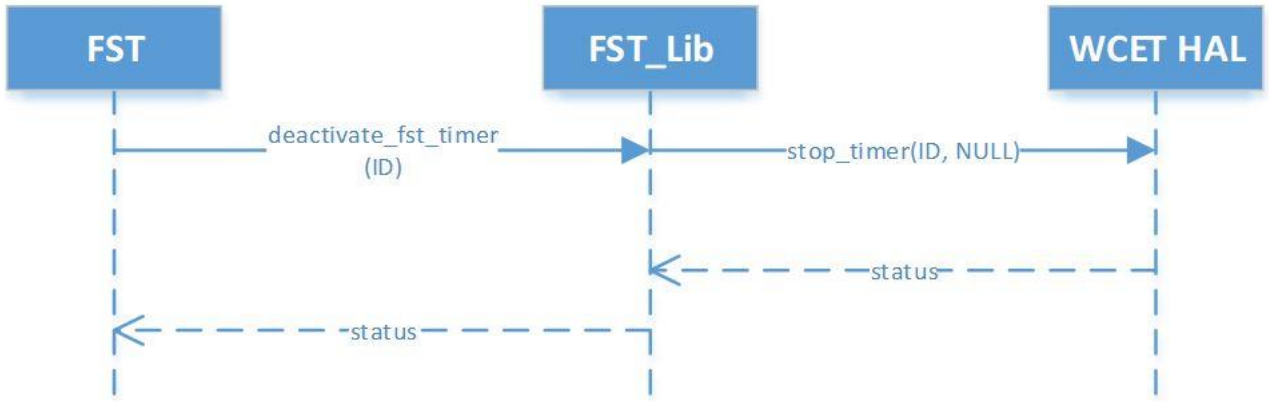


Figure 14 Delete FST Timer

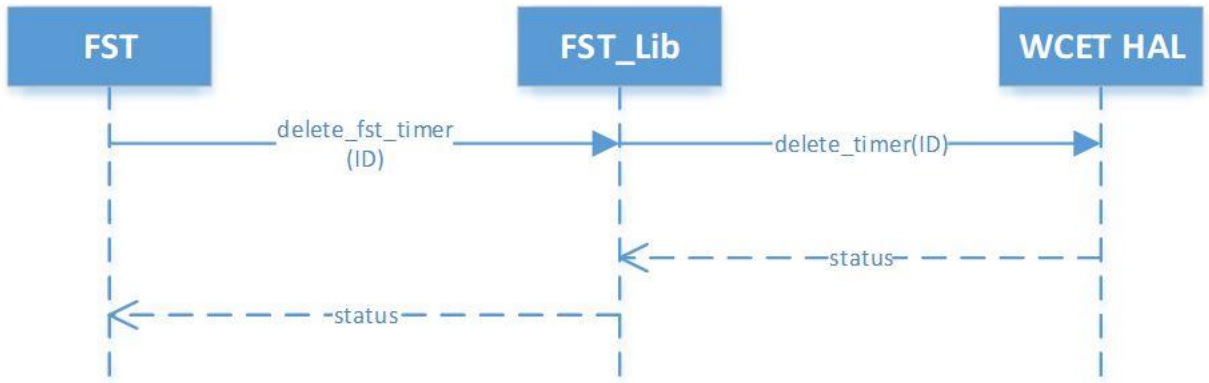
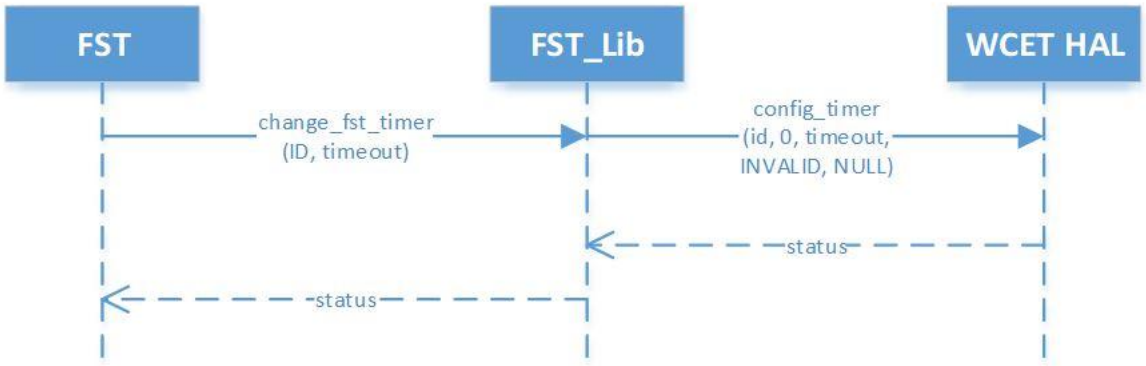


Figure 15 Change FST Timer



3.3.2 Interrupt Handling

ARM Cortex M3 provides an inbuilt Nested Vectored Interrupt Controller (NVIC). NVIC can support up to 240 external interrupts. However, for SCI, HW is going to enable only 64 external interrupts.

NVIC supports 256 priority levels for interrupts that can be dynamically re-prioritized. Vector Table contains the address of the corresponding handlers and the base address of the Vector table can be programmed in the appropriate register.

All the drivers that involve interrupt handling, register their corresponding interrupt handlers in the Vector Table. When any external interrupt occurs, the NVIC routes it to the appropriate handler as programmed in the Vector Table.

The ISR will do minimum processing and schedule a thread for actual interrupt processing. This is like top-half/bottom-half implementation of ISR, but one half is ISR context within ISR. Other half i.e. actual task to be performed on receiving the interrupt is done in the thread context that is scheduled by ISR using RTOS inter thread communication mechanisms. This thread will get more information about the interrupt including sending message to appropriate destination and then send it to the right thread/task/FST.

Each external interface driver will have associated upper/hardware independent layer. Callback function in this layer is invoked by ISR. The callback function shall signal some event to be dispatched which will invoke thread that should take over from the IRQ handler of that driver.

Each Interrupt will be assigned a priority depending on the criticality of the interrupt. If the interrupts are assigned same priority in the NVIC, their priority is determined by the IRQ line number to which they are connected.

Following table shows the various interrupts and related information. The priority of each of the interrupt will be decided during implementation phase and based on various performance analysis. The priority information will be documented in MDS.

Table 5 List of Interrupts

Interrupt name	Number of Interrupts	Description
FMM Error	1	Interrupt for FMM Error
FMM Warning	1	Interrupt for Alarms configured as warning
FMM Info	1	Interrupt for Alarms configured as Info
SPI Slave- for communication with safety MCU/other SCI	1	
SPI Master- for communication with other SCI	1	
Mailbox – for communication with Host using PCIE MB and with PCode using RAVDM	1	
PMC IPC-for communication with PMC firmware	1	
CSE IPC-for communication with CSE firmware	1	
SB Pkt Gen P completion interrupt	1	
SB Pkt Gen NP Completion Interrupt	1	

Global PMC Reset Req Interrupt	1	
Early Reset Warning Interrupt	1	
GPIO	1	Single interrupt for all GPIOs. Status register can be checked in ISR to indicate which GPIO is asserted
GP Timers Expiry	12	GP Timer Expiry Interrupt
Periodic Timer Expiry	1	GP Timer Expiry Interrupt

3.3.3 Exception Handling

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_843062	Accepted		SIL 3	1. Entry 6 (Usage Fault) in interrupt vector table is configured with internal fault handler. If CPU accesses any location that contains invalid opcode, it will jump to Entry 6 (Usage Fault) in interrupt vector table and call internal fault handler from there.		N/A	Test	Intel Confidential
SSA_843063	Accepted		SIL 3	2. Entry 4 (Memory management Fault) in interrupt vector table is configured with internal fault handler. In case of any MPU violation, it will jump to Entry 4 (Memory management Fault) in interrupt vector table and call internal fault handler from there		N/A	Test	Intel Confidential
SSA_843065	Accepted		SIL 3	3. All interrupts not used by firmware (unexpected interrupts) shall also be treated like exceptions and entries for them in NVIC vector table shall be configured with internal fault handler. The internal fault handler will cause NOK to be asserted and required diagnostic data logging flow shall follow.		N/A	Test	Intel Confidential

3.3.4 Thread Priority

Table 6 List of threads and priorities

S.No.	Thread Name	Thread Priority (lower the value, higher the priority)
1	Error 2/fatal FST	1
2	rst_req_svc_thread	16
3	Proof Test Service	16
5	MCU Service	16
6	FSTM Main Thread	6
7	MB domain Thread	16

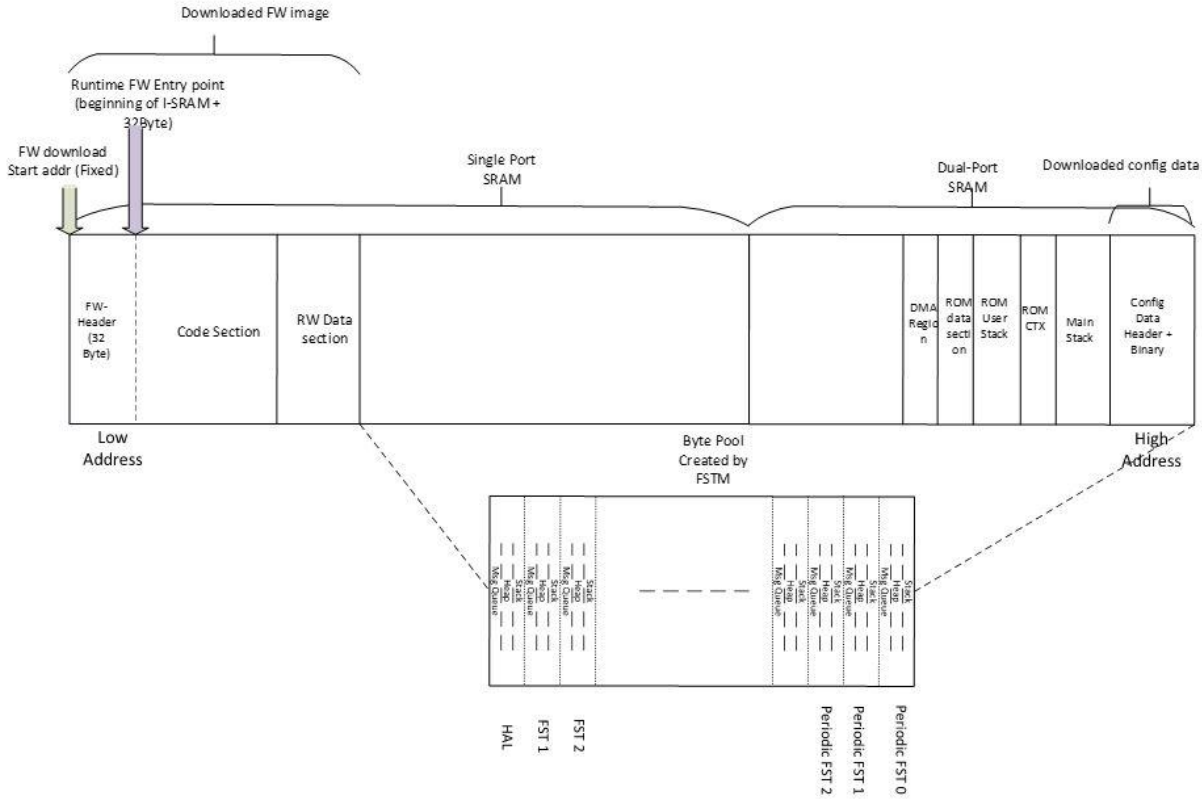
S.No.	Thread Name	Thread Priority (lower the value, higher the priority)
8	SPI domain Thread	16
9	RAVDM domain Thread	16
10	Timer Service Timer Expiry handler	2
11	Timer Service Host Timer handler	16
12	Logger worker thread	16
13	CSE interface thread	16
14	Mailbox async thread	16
15	SPI async thread	16
16	GPIO async thread	16
17	Boot FST	16
18	PLL, VID and thermal FST	16
19	ODCC	16
20	SCI Internal FST	16
21	frCPU Thread	16
22	Host STL	16
24	UCNF FST	8
25	correted error FST	16

3.3.5 Memory Layout

In SCI, the SRAM is divided into a 128 KB dual-port SRAM and a 384KB single port SRAM.

Based on the layout of these two instances of SRAM, the Memory layout of the SCI FW and data would be as follows:

Figure 16 Memory Layout



The SCI ROM uses SRAM regions as shown above for its data sections and stack usage. It also downloads the SCI FW and SCI Config images in the regions as shown above.

However, once the SCI FW starts running, can reclaim the entire memory associated with SCI ROM and even config data (after it has consumed it).

The FSTM is provided with the start address of the unused memory as shown in the above diagram. It then creates a byte pool with this and size till the end of the Dual port SRAM, thus reclaiming the memory used by the ROM data sections, stack and config data (after FSTM has consumed it). This Byte pool is then used for all the memory allocation used for the HAL, FSTs etc.

For more details on ThreadX Byte pool refer ThreadX user guide.

3.3.6 Diagnostic data Logging Service

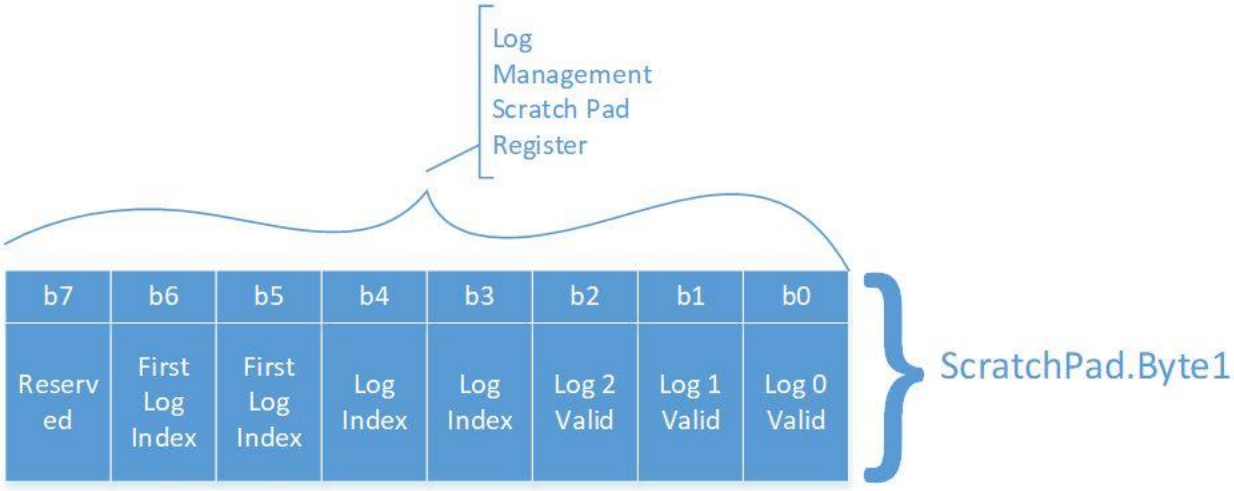
Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 590296	Accepted		SIL 3	SCI does not have direct access to flash and hence this service will be changed so that it works on top of existing EHL (and other Intel platform) features. For SCI, the diagnostic data logging is a best effort capability where we use PMC and host SW to do the actual diagnostic data logging. Whenever ISI FW detects a Warning or Fatal error, it tries to log the diagnostic data associated with the event		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				in the PMC DSW. The high level flow is as follow: - SCI uses PMC IOSF-SB MEMWR operation to store 32 bytes diagnostic data log in PMC managed register file powered by DSW. This rail will not lose power during most resets (except G3). * Amount of RF space is limited to 128 bytes. So at most 3 diagnostic data logs can be maintained till next reset (32 bytes allocated for additional control information) - SoC reset takes place due to any reason.				
SSA_ 843067	Accepted		SIL 3	- Upon next boot, host BIOS/FSP/OS application sends command to SCI using PCIe mailbox to read all available diagnostic data logs. Therefore, * Data is not updated as soon as there is an error detected (even correctable / non fatal). * Data is not updated in any periodic manner. - SCI sends IPC to PMC to read all the diagnostic data logs in the RF space - SCI returns diagnostic data logs to OS SW which will persist in OS managed storage.		N/A	Test	Intel Confidential

3.3.6.1 PMC Diagnostic data Log Management

The PMC has 128 (32 Bytes * 4) bytes of scratch pad space which can be used to store the diagnostic data logs and ISI state. The first 32 bytes are used to save the state of ISI across ISI reboots. The remaining (32 * 3) bytes are used to store last three error diagnostic data logs. The Byte30 and Byte31 are used to store 16bit CRC (CRC16-CCITT with initial value of 0xFFFF) and this CRC will be computed by FST that will be storing the error diagnostic data log in PMC DSW area. Only the first 30 bytes are considered as safety component and hence CRC is computed only for first 30 bytes. The remaining (32 * 3) bytes used to store the diagnostic data logs is not considered as safety components.

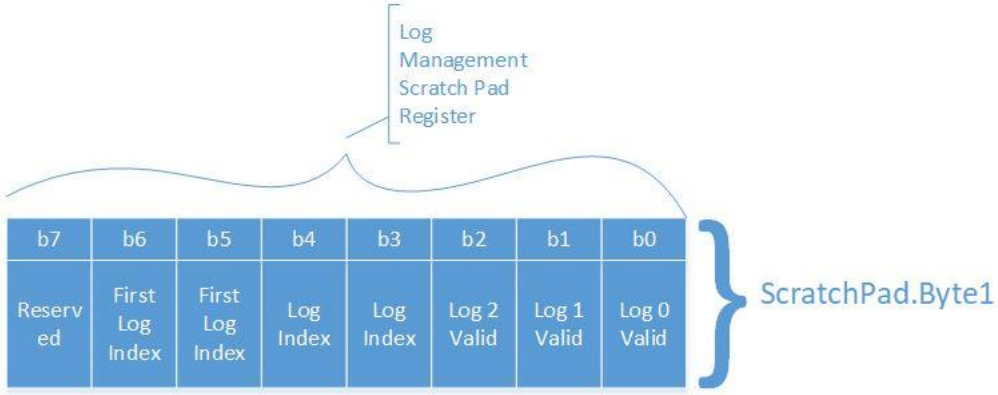
Figure 17 PMC Diagnostic data Log Management



The first byte of scratch pad is used for 1oo2D "Proof Test" to store the state of ISI during multiple ISI reboots. The details of the first bytes can be found in 1oo2D proof test flow.

The second byte of scratch pad is used to store the state of diagnostic data log. The first three bits will tell whether the diagnostic data log is valid and the next two bits will tell the Index of diagnostic data log. The details of how these bits will be used to write/read the diagnostic data logs to/from the PMC scratchpad area can be found in write diagnostic data log and read diagnostic data log sections.

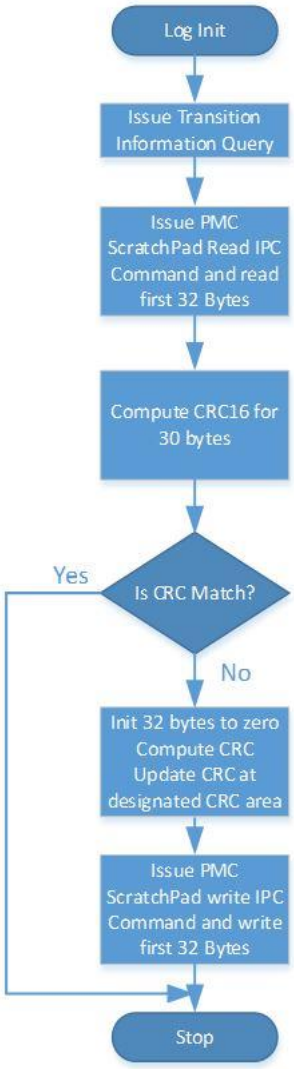
Figure 18 Diagnostic data Log Management Scratchpad Register format



3.3.6.1.1 PMC Diagnostic data Log Init

During first boot the CRC16 must be initialized. Logging Init API will read the first 32bytes of PMC scratchpad and recompute the CRC16 for first 30 bytes. If the CRC matches with CRC stored in byte 30 and byte 31 then the CRC initialization will be skipped, else all 128 bytes of scratchpad register are initialized to zero and CRC is also initialized.

Figure 19 CRC Init if DSW is invalid

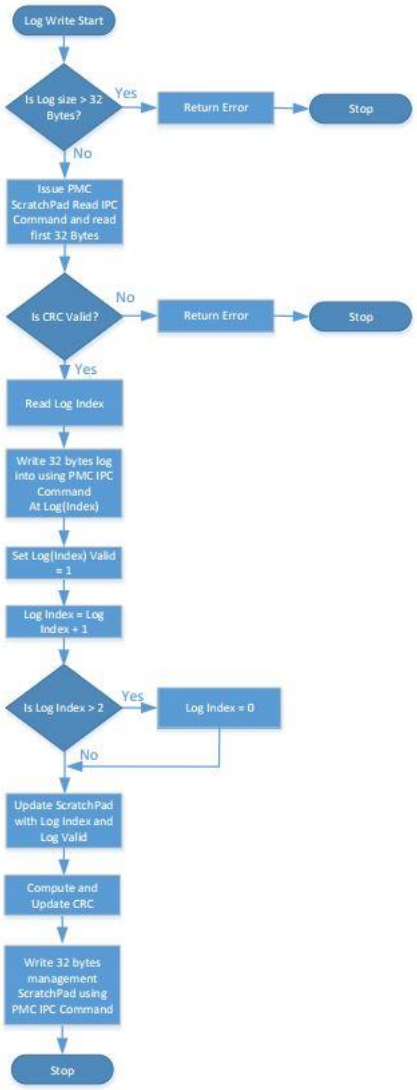


During boot, SCI will check if ISI_128_DSW_VALID is True. If it is False, then SCI will consider the diagnostic data log scratchpad as invalid. SCI will init the scratchpad area to zero and update CRC at CRC designated area. SCI will write the updated scratchpad area to PMC.

3.3.6.1.2 PMC Diagnostic data Log Write

When there is a FATAL error condition, the error log cannot be forwarded to host and a temporary storage is required. ISI will call the PMC Log Write interface to store the logs in PMC scratch pad area.

Figure 20 Flow chart for PMC Diagnostic data Log write



1. PMC diagnostic data Log write function will check whether the incoming diagnostic data log size is great than 32 bytes. If greater than 32 bytes, then the function will return error
2. PMC diagnostic data Log write function will issue PMC IPC command to read first 32 bytes from PMC ScratchPad
3. Recompute the CRC and validate it with the CRC present in the scratchpad. If the CRC does not match, then return error.
4. PMC diagnostic data Log write function will write the diagnostic data log to PMC Scratchpad area at diagnostic data Log(Index)
5. PMC diagnostic data Log write function will set diagnostic data Log(Index) Valid = 1

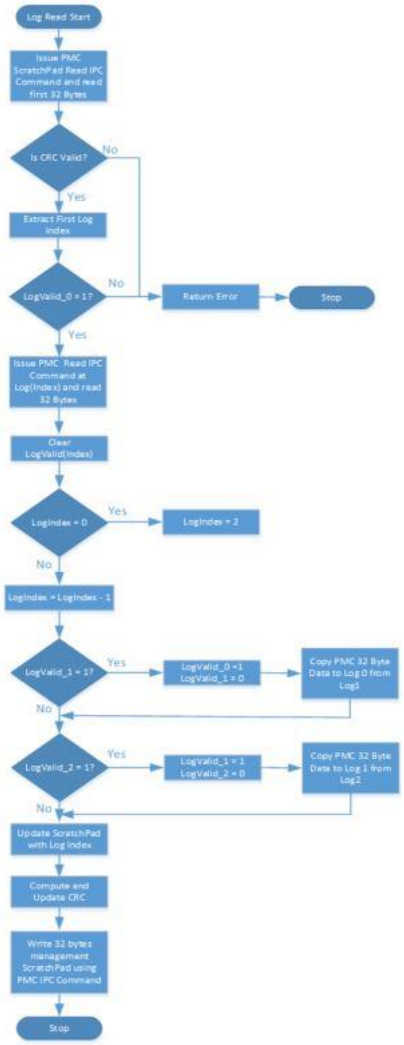
6. Increment the diagnostic data LogIndex by 1. If diagnostic data LogIndex > 2 then Reset diagnostic data LogIndex = 0 and then update the new diagnostic data Log Index. This step is necessary to read the stored diagnostic data logs as FIFO
7. Update diagnostic data Log Index and update CRC
8. Issue PMC IOSF-SB MEMWR operation and write the management scratchpad.

3.3.6.1.3 PMC Diagnostic data Log Read

Whenever there is a FATAL error, error diagnostic data logs will be stored temporally. On next boot host will request the diagnostic data log using mailbox command. ISI firmware will initiate PMC Read Diagnostic data Log and forward the diagnostic data log to host. Whenever a diagnostic data log is read, it will be marked as deleted by setting LogValid = 0. Following steps are followed during PMC Diagnostic data Log read command flow.

1. PMC diagnostic data Log Read will read the scratchpad area
2. Validate CRC. If CRC mismatch return the error.
3. Check if the first diagnostic data log if valid. If first diagnostic data not valid then, return error
4. PMC diagnostic data Log Read will extract diagnostic data Log Index from local copy of ScratchPad.Byte1
5. If Index = 0 then set Index = 2 else decrement Index by 1
6. Issue PMC diagnostic data Log read of 32 byes at diagnostic data Log(Index)
7. If LogValid_1 = 1, then Copy the 32 bytes of diagnostic data from LogIndex_1 to LogIndex_0
8. If LogValid_2 = 1, then copy the 32 bytes of diagnostic data from LogIndex_2 to LogIndex_1
9. Recompute CRC and update scratchpad
10. Write updated scratchpad to PMC

Figure 21 PMC Diagnostic data Log Read flow



3.3.6.2 System Event Log format

The diagnostic data log format will be same between SNI and SCI with changes to sensor numbers and channel numbers. The same diagnostic data log format is used for host logs, PMC diagnostic data logs and other formats. The format of diagnostic data logging will be as follow.

Table 7 Diagnostic data logging format

Byte	Field	Description
1, 2	Record ID	Record ID will uniquely identify each Event Log. Record ID values 0000h and FFFFh are reserved. Whenever Record ID reaches FFFEh, the Record ID will be restarted from 0001h
3	Record Type	02h – SCI Event Record

		C0h-DFh – OEM timestamped Event Record E0h-FFh – OEM non-timestamped Event Record
4, 5, 6, 7	Timestamp	Time when event occurred. LS byte first
8,9	Generator ID	This is a combination of EventLog Source and Channel Number. Byte 1: [7:1] – 7 bit System Software ID or SPI slave address [0] : 0b – ID is SPI slave address 1b – ID is system software ID Byte 2: [7:4]: Channel Number. It is the channel number on which the event was received [3:0]: reserved. Set to be 0000b
10	Event Log Version	04h for this specification
11	Sensor Type	Sensor Type Code for the sensor that generated the event
12	Sensor #	Sensor number that generated the event
13	Event Direction | Event Type	Event Direction [7] – 0b = Assertion Event 1b = Deassertion event Event Type : Refer IPMI spec 42.1 for Event types
14	Event Data 1	Event Data will change with every log format
15	Event Data 2	Event Data will change with every log format
16	Event Data 3	Event Data will change with every log format
17~30	Extended Event Data 1~15	Extended Event Data will change with every log format
31~32	CRC16-CCITT	CRC-16 computed on first 30 bytes, with initial value of 0xFFFF

3.3.7 Drivers Subsystem

SCI FW infrastructure (for EHL platform) will include various drivers for peripherals and interfaces. The complete list is as follows:

- PCIe subsystem driver (covers PCIe MB and RAVDM)
- SPI slave
- SPI master

- GPIO
- Timers
- fRCPU
- FMM
- IPC to CSE and PMC
- DTF
- Side-Band Packet Generator
- Watchdog
- Local Control

3.3.7.1 PCIe subsystem driver (covers PCIe MB and RAVDM)

The PCIe subsystem driver available in SCI FW is responsible for managing the PCIe MB IP Block, which acts as an interface to all communication w.r.t. EHL CPU. The following are the major responsibilities for the PCIe subsystem driver.

1. PCIe mailbox based host communication – The PCIe mailbox based host communication provides support for communication to the software executing on host CPUs which includes BIOS, workloads etc. The PCIe mailbox based host communication is covered in detail in Section “Host communication over PCIe Mailbox” below.

2. RAVDM based communication – The RAVDM based communication mechanism provides support communicating with the PUnit controller, which is a part of EHL CPU. The RAVDM based communication mechanism is discussed in detail in Section “PUnit communication over RAVDM”.

3.3.7.2 SPI Slave driver

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590320	Accepted		SIL 3	SPI slave driver is responsible for managing all activities occurring with respect to SPI slave IP. Following are the major responsibilities of SPI slave driver: • Initialize and configure the SPI slave device • Receive data over SPI interface from an SPI master (typically external MCU in EHL use case) and respond to the requests • Error handling during transfers		N/A	Test	Intel Confidential

3.3.7.3 SPI slave and master driver

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590322	Accepted		SIL 3	The SPI slave and master driver are used to manage the SPI master and slave IPs, which is used primarily for inter-SCI communication in 1oo2D use cases. The SPI master of one SCI channel is connected to the SPI slave of another channel and vice-versa. Following are the major responsibilities of this driver: • Initialize and configure the SPI slave as		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				receiver and SPI master as transmitter • Receive messages over SPI slave and restrict transmission over SPI slave at all times • Transmit messages over SPI master and restrict receiving messages over SPI master at all times • Error handling during transfers				

3.3.7.4 GPIO Driver

GPIO driver provides functionality to request for notification on GPIO pin's value change.
It provides two mechanisms for this:

- Wait for a PIO event, which will block the execution of the caller, until the event occurs.
- Register a callback function, which will be called when PIO event occurs.
  GPIO driver provides functionality to register multiple callback functions for the event on the same pin or register a single callback function to be registered for multiple pins.

GPIO driver also provides mechanism to write a value to GPIO pin.

3.3.7.5 DMA driver subsystem

The DMA subsystem includes a dual lock-step DMA engine using which data can be copied between memories and/or peripherals. The DMA core controller doesn't support scatter-gather based memory copy. The following are the major responsibilities for the DMA driver subsystem.

- **Memory buffer copy** – The DMA driver subsystem provides a mechanism to copy memory buffers from one address region to other.
- **Error handling** – The DMA driver subsystem tracks errors occurring during memory copy and reports the same to caller for retry operation or to employ alternate software based memory copy

Following are the end points (for both Read and Write) associated with DMA subsystem:

1. Internal SCI SRAM
2. CSE SRAM
3. Host DDR

The DMA subsystem used in SCI is used to transfer large sized memory blocks between hosts to SCI for latency sensitive applications.

3.3.7.6 Timer driver

This driver exposes interfaces to modify various registers associated with GP Timers, Periodic Timers and to read counter values corresponding to RTOS timer. It will also provide required interface for error injection.

3.3.7.7 frCPU driver

In SCI, all the fRCPU alarms are routed to the FMM. Hence, the fRCPU driver need not register its own ISR. It should provide interfaces that perform the following functionality:

- Interface to initialize and setup fRCPU
- Interface to Read the Control and Status registers to find details on the error that fRCPU has determined.

3.3.7.8 FMM driver

FMM driver provides the following features.

- Initialization of FMM and registration of FMM alarms with NVIC.
- Configuration of alarm severity levels for the different alarm signals of CPU, PCH, and internal SCI errors etc.
- Masking of alarms required for some boot time tests. This is required to prevent NOK assertion for faults injected by boot time tests.
- Unmasking of alarms after execution of boot time tests.
- For some alarms, default state is masked because at startup, before firmware comes up, these alarms need to be ignored. FMM shall provide API to selectively unmask selected alarms.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 851608	Accepted		SIL 3	Generation of OK/NOK/alert signals based on FSTs/FSTM request. While generating OK, driver shall check that previous state is not NOK. If previous state is NOK, driver shall return error and transition to OK state shall not take place.		N/A	Test	Intel Confidential
SSA_ 851609	Accepted		SIL 3	After generating of OK/NOK/alert signals, FMM driver need to do loop back test for OK/NOK/alert signals before returning the status.		N/A	Test	Intel Confidential

- Handling alarm interrupts.
- Reading of required status registers for error isolation.
- Any FMM API shall unlock FMM global configuration before making any changes in FMM and lock it before returning. It shall not be left unlocked.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 863475	Accepted		SIL 3	Any RW FMM register written should be read back. If read contents are not found to be same as written, FMM API shall return error.		N/A	Test	Intel Confidential

3.3.7.9 IPC to CSE and PMC driver

This driver module provides mechanism to perform IPC handshake with CSE and PMC and interfaces for FSTs/FSTM to send and receive IPC messages to and from CSE and PMC firmware.

With PMC, ISI will be the IPC-Leader and with CSE, it will assume the Follower role.

3.3.7.10 SB packet generator driver

The driver will allow sending posted or non-posted transactions.

3.3.7.11 DTF driver

DTF driver provides a mechanism to FSTs/FSTM for sending diagnostic data log messages over DTF interface to the North Peak Fabric. The DTF interface is used in place of UART interface in case of integrated SCI solution.

3.3.7.12 Watchdog driver

This driver provides interfaces to initialize, configure, start, and reload the WDT. Any RW register write by driver shall be read back. If read contents are not same as written, API shall return error.

3.3.7.13 Local Control Driver

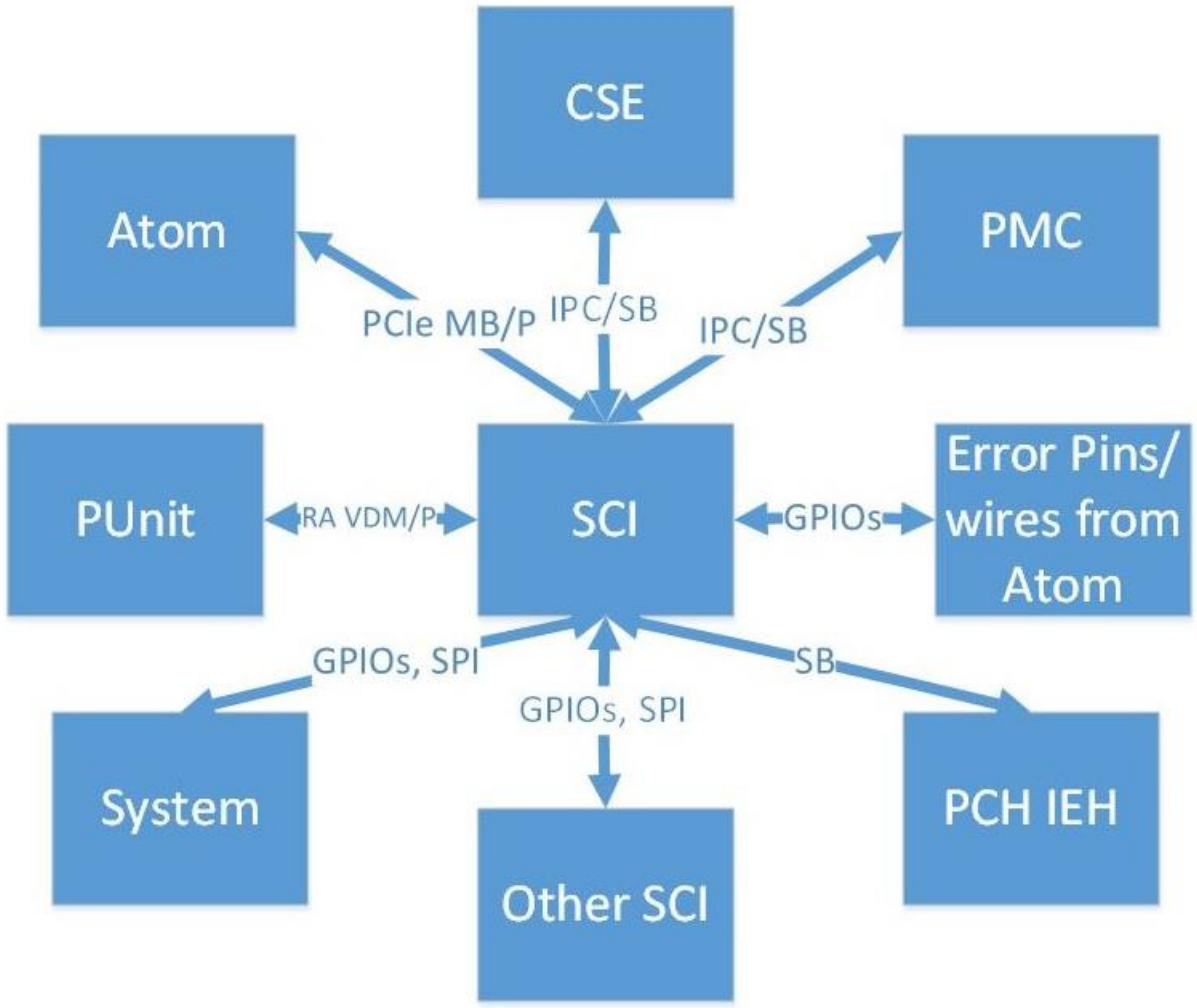
This driver handles interrupts for signals coming from PMC for early reset warning and reset prep. The driver also provides interface to configure registers in local control e.g. configure clock monitoring, reset request to PMC etc.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_848945	Accepted		SIL 3	Any RW register shall be read back by the driver writes into it. If read contents are not same as written, API shall return error.		N/A	Test	Intel Confidential

3.4 Communication Mechanisms

Following diagram shows the various entities inside and outside the PCH with which the SCI will interact:

Figure 22 SCI Interactions



1. **CSE**: SCI will interact with CSE over IPC (going over IOSF SB) for following use cases:

    a. Boot flow: SCI will boot from ROM; the runtime FW will however be loaded by CSE from flash memory as part of the boot flow. CSE will be responsible for loading the FW image (along with configuration data), verifying the integrity as part of secure boot and then indicating to SCI so the FW can be copied to SCI SRAM. Refer to Boot section for further details.

    b. Proof test orchestration: As CSE in MCC is inherited from TigerLake PCH, the Proof test handling is part of CSE FW. SCI and CSE will interact for Proof test results sharing. After completion, CSE will send a result (pass or fail) to SCI over IPC.

2. **PMC**: SCI and PMC interaction (using IPC over IOSF SB and local control interface for reset) includes the following use cases:

    a. Diagnostic data Logging. See Diagnostic data logging section for details.

    b. IPC for platform transition information query from SCI to PMC.

c. Early reset indication. For most of the reset scenarios (exceptions include critical ones), PMC will inform SCI for upcoming reset through early reset warning and reset prep request. Both inputs come to SCI through interrupts in local control. SCI will perform required book keeping like stopping various FSTs before giving an ok to PMC. To ensure faults in SCI do not stop the reset for a longer time, PMC will use a watchdog timer.

3. **Host/Atom**: There are primarily following use cases for SCI to Host communication (using PCIe Mailbox over IOSF Primary) currently being planned:

   a. ODCC/LCLS: On demand cross comparison/Loosely coupled lock step on host. In this case SCI is used to compare the results associated with the workloads running on two host cores.

   b. Various STLs communicating status/results to SCI.

   c. WCET/BCET: Worst and best case execution timer.

   d. Diagnostic data Logging: Host will get the diagnostic data logs saved from the prior boot using this interface.

4. **PUnit**: SCI to PUnit communication is primarily needed for getting Voltage, DTS, PLL lock status values and reading MC Banks.

5. **Various external wires and pins**: PCH and Compute errors signals are connected to SCI over various wires and pins.

6. **Other SCI**: For 1oo2D SIL-3 setup, we will have two EHL channels working together to provide ODCC like setup where two SCI instances will interact using SPI. We will have two separate SPI channels for this communication so that each of the two SCIs can initiate connection on its own. The OK/NOK of the SCIs are interconnected. The OK/NOK of SCI1 is connected to SCI2 and vice-versa.

7. **PCH IEH:** IEH is connected to SCI over SB-Packet Generator. IEH has error information for errors in the platform. SCI uses IOSF SB driver to do a posted read to read IEH registers to know the reason of errors. SCI provides this information in error logs.

8. **System**: There are two scenarios for SCI to communicate to outside world:

   a. When external world includes an intelligent MCU. In this case the communication will be over 3 Pin OK/NOK/Alert interface and SPI interface where SCI will act as a slave. This slave interface is used by the external MCU to send additional commands to SCI.

   b. When external world is “dumb” made up of various relays and switches (but no MCU). In this case SCI will interact over the 3 Pin OK/NOK/Alert interface.

3.4.1 Generic requirements for communication mechanism

All communication mechanisms are required to provide end-to-end safe interfaces for sending and receiving messages. Firmware will implement end to end safety protocol for all the communication interfaces mentioned below.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_971267	Accepted		SIL 3	1. Host communication over PCIe Mailbox 2. PUnit communication over RAVDM 3. External MCU communication over SPI 4. SCI-SCI communication over SPI 5. PMC communication over IPC register space Note: In case of PMC communication over		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				IPC register space, a CRC field is maintained in RF data space which helps in providing the message integrity.				

Other communication interfaces such as CSE communication, IEH communication and external wires and pin interfaces are operated without or limited end to end safety mechanism.

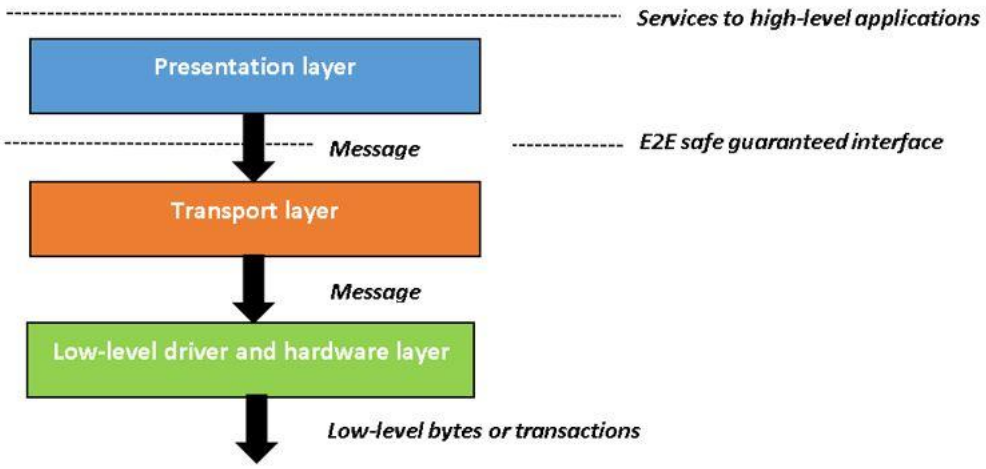
The communication protocol will include the following features to achieve the end-to-end message transfer ability.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_971269	Accepted		SIL 3	• Message integrity check on data payload – Can be CRC8 or CRC16 or CRC32 depending on the size of message. SCI FW will assert NOK if it finds a mismatch		N/A	Test	Intel Confidential
SSA_971270	Accepted		SIL 3	• Message identification with unique ID or sequence number. SCI FW will assert NOK if it finds a mismatch • Retry functionalities or NOK Assertion in case of message timeout.		N/A	Test	Intel Confidential
SSA_972010	Accepted		SIL 3	• Message source and destination address check. SCI FW will drop the message if it finds a mismatch		N/A	Test	Intel Confidential

3.4.2 Layered architecture of communication stack

The communication stack of SCI firmware is organized into 3 layers as shown in Figure 23. This layered architecture is similar to the OSI model used in computer networks but is much simplified.

Figure 23 Generic layered architecture diagram for communication stack



The details of the layers are as given below.

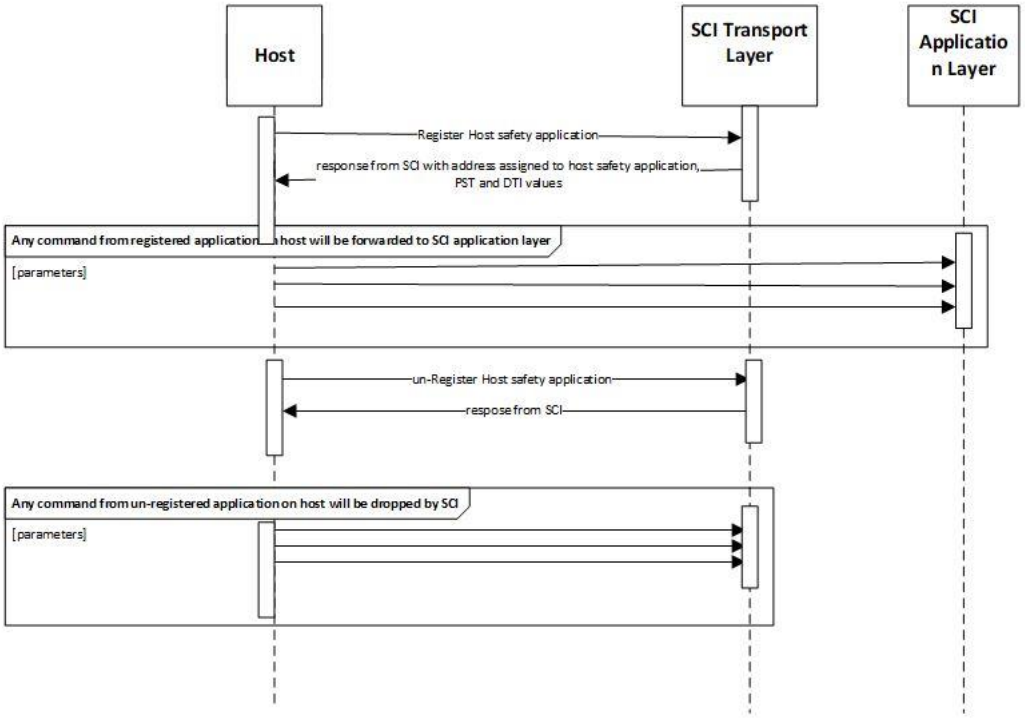
- Presentation layer provides services to high-level applications - FSTM and other FSTs. This layer converts the high-level application services to messages, thus abstracting them from applications. The presentation layer

outputs a message to the transport layer. It is to be noted that the timeout and retry functionalities which are key to achieve the end-to-end safe communication are provided in this layer or at application layer (by FSTs and FSTM). This layer could be optional in some protocols in which case the message formatting and retry-timeout functionalities should be covered by either FSTs or FSTM in the application layer.

- Transport layer provides mechanism for reliable end-to-end safe delivery by adding CRC protection, sequence number and request-response matching. It also assigns address to applications (applicable only for Mailbox interface) when it receives registration command from an application. After that, it treats only those applications as authorized applications. If any unauthorized application tries to send any command to SCI, it gets dropped by transport layer. Along with registered applications, transport layer treats certain default applications like BIOS as authorized application. List of authorized applications that can interact with applications on SCI are maintained in transport layer. The transport layer outputs a message to the low-level driver (OSDL part as explained below). It is to be noted that the retry-timeout functionalities are not a part of this layer.

Host Safety Application registration:

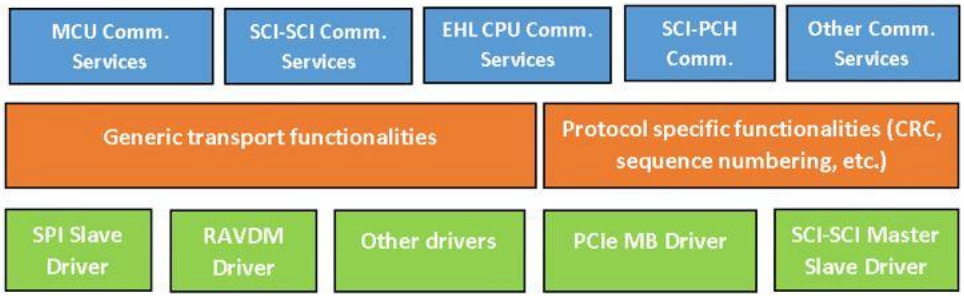
Figure 24 Host Safety Application Registration



• Low-level driver layer is the raw low-level driver which deals at a byte level transfer (or groups of bytes or raw transactions) across the hardware interface. This layer is agnostic to message formats. There are two parts of this layer – operating system dependent layer (OSDL) and operating system independent layer (OSIL). OSDL layer is dependent of the OS, ThreadX in this case and OSIL is kept free of any OS constructs for portability reasons. This layer transmits and receives the messages as a series of byte transfer or transactions.

An implementation oriented layered diagram for SCI is provided in below figure.

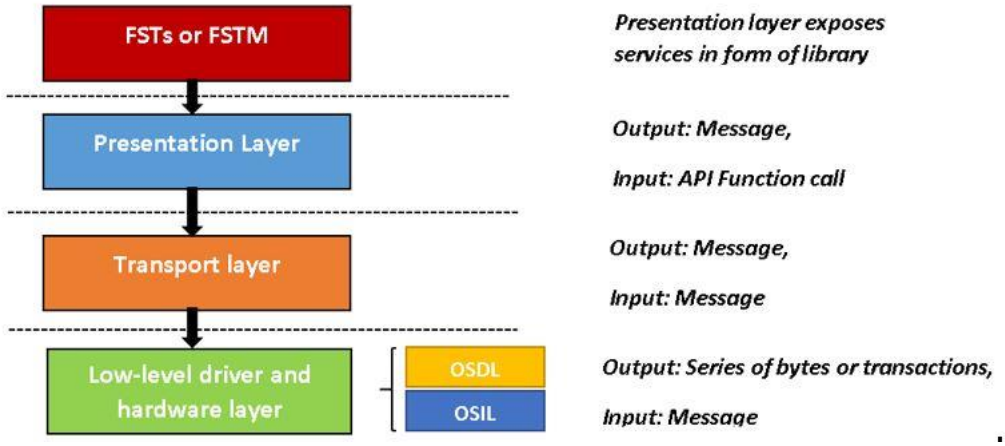
Figure 25 SCI's architecture diagram for communication interfaces



3.4.2.1 Interface interaction points in layered architecture

The interaction points across the interfaces in the 3 layer architecture is provided in figure below.

Figure 26 Interfaces and interaction points in 3-layer architecture



The interaction points between the layers are based on messages. Messages represent the basic unit of end-to-end safe data transfer between SCI firmware and other software entities such as Safety MCU firmware, BIOS, OS and PCU firmware. Thus, messages contain placeholders for CRC, sequential number and other transport control flags. A message may be split in multiple transfers or transactions based on the underlying hardware layer. The processing and queueing of messages across the layers is based on first-come first-serve (FCFS) policy.

The interface between transport layer to driver layer's OSDL is based on messages. The interface between application layer presentation layers is based on API functions. The interface between presentation layer and transport layer is based on API functions and is defined as follows,

- return status, transaction ID = send(endpoint address, outgoing request)
- return status, incoming response = receive(endpoint address, msgType msg, timeout)
- return status = cancel(endpoint address, msgType msg)
- return status, transaction ID, incoming request = listen(endpoint address, timeout)
- return status = respond(endpoint address, msgType msg)

3.4.2.2 Interface specific requirements, details and message formats

The interface specific details of the various communication mechanism in SCI are discussed in the following section.

3.4.2.2.1 Supported Safety mechanism for various communication channels

Below table shows the various communication channels and the corresponding safety mechanisms enforced for those channels.

Comms Channel → Safety mech	Seq No	Timestamp	Time expectation	Connection authentication	Feedback message	Data Integrity	Redundancy with cross checking	Different data integrity assurance
ISI to Host	X			X	X	X		
ISI to PMC([1])						X		
ISI to PUnit	X				X	X		
ISI to ISI	X			X	X	X		
ISI to Ext MCU	X			X	X	X		

- ([1]) Since communication with PMC is only used to store the logs and during proof test, the communication link only maintains a timing expectation (Timeout) and a CRC for data integrit

3.4.2.2.2 Host communication over PCIe Mailbox

The details of the messaging interface for the host and SCI communication over PCIe MB are as follows.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_971271	Accepted		SIL 3	Support for bi-directional messaging channel between host software and SCI. Both host software and SCI act can send requests to other side as well as process incoming requests and respond. Refer to Section "PCIe MB based services" for details on list of services supported over this interface.		N/A	Test	Intel Confidential
SSA_971272	Accepted		SIL 3	Each PCIe MB carries up to 128 bytes of data and support for fragmentation is not planned.		N/A	Test	Intel Confidential
SSA_971273	Accepted		SIL 3	All PCIe MB messages are protected by CRC-16 (CRC-CCITT CRC polynomial).		N/A	Test	Intel Confidential
SSA_971274	Accepted		SIL 3	Supports an 8-bit sequence number in command and response packets. Command/response packets must be matched using sequence number.		N/A	Test	Intel Confidential
SSA_971275	Accepted		SIL 3	Supports source and destination addresses. Source address in incoming command not known/registered with SCI, is not serviced by SCI. Command from known /registered address is passed to upper layers of software in SCI, else error is passed as return code of APIs. SCI FW upper layers will copy the incoming source address field to destination address field in response packet, else the command is simply dropped.		N/A	Test	Intel Confidential

3.4.2.2.3 Safety Workload Registration with SCI

1. Command for registration of workloads from host will terminate in transport layer, so transport layer will assign addresses to safety workloads and provide it back in response and store this list of valid addresses that can talk to SCI.

2. In this list of valid addresses will also be some hard coded addresses which include address coming in command from BIOS and POSC test result(they do not specifically register with SCI)

3. Applications/FSTs do not get any information about it.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_967630	Accepted		SIL 3	When any command from unknown address comes to SCI, transport layer does not forward it anywhere. So sender does not get any response or any action from SCI. This ensures that SCI transport layer allows only authorized entities to talk to SCI applications.		N/A	Test	Intel Confidential

3.4.2.2.4 PUnit communication over RAVDM

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590363	Accepted		SIL 3	The details of the messaging interface for the SCI and PUnit communication over RAVDM are as follows. EHL has extended the existing DSKU/CSME mailbox backbone to support the SCI2PCU HW/FW mailbox interface. The details of this communication channel are as follows. 1. SCI acts as a requestor for all communication between SCI and PUnit controller. PUnit controller acts as responder in all use cases. Refer to Section "SCI – PUnit commands" for details on list of services supported over this interface. 2. Only one outstanding request from SCI is allowed at all times. 3. Request messages from SCI to PUnit can be up to 16 bytes and responses messages from PUnit can be up to 144 bytes. 4. Request messages from SCI to PUnit are protected by CRC-8(0x1D polynomial) and response messages from PUnit to SCI are protected by CRC-16(CRC-CCITT CRC polynomial). 5. Support for up to 4-bit sequence numbers but no support for fragmentation due to interface limitations. 6. Both request and response contain 8 bit command number and 4 bit sequence number. Both command number and sequence number are compared in response message		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				from Punit to match the response to sent command.And if a mismatch if found, NOK is asserted.				

3.4.2.2.5 External MCU communication

This section is applicable to 1oo1 smart configurations, where in SCI communicates with the intelligent MCU as part of external world (also known as external MCU). The details of the messaging interface for SCI-intelligent MCU communication are as follows.

1. SCI acts as a responder for all communications between SCI and external MCU. External MCU acts as a requestor for all communications between SCI and external MCU. Refer to Section "SCI - External MCU communication" for details on list of services supported over this interface.
2. External MCU-SCI communication messages can be up to 512 bytes of data.
3. External MCU-SCI communication messages are protected by CRC-16 (CRC-CCITT CRC polynomial).

3.4.2.2.6 SCI-SCI communication

This section is applicable to 1oo2D cross monitoring configurations only in which we have two SCIs communicating with each other over SPI links. The details of the messaging interface for SCI-SCI communication are as follows

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_971276	Accepted		SIL 3	Support for bi-directional messaging channel between the two SCIs. It is possible for one SCI to act as a requestor as well as a responder simultaneously.		N/A	Test	Intel Confidential
SSA_971277	Accepted		SIL 3	Each SCI-SCI message can carry up to 512 bytes of data and support for fragmentation is not planned.		N/A	Test	Intel Confidential
SSA_971278	Accepted		SIL 3	All SCI-SCI messages are protected by CRC-16 (CRC-CCITT CRC polynomial).		N/A	Test	Intel Confidential
SSA_971279	Accepted		SIL 3	Support for a 32-bit timestamps (acting as nonce) is provided. Back-to-back messages with same nonce is treated as stuck-at fault condition (communication error).		N/A	Test	Intel Confidential
SSA_971280	Accepted		SIL 3	Supports an 8-bit sequence number in command and response packets. Command/response packets must be matched using sequence number.		N/A	Test	Intel Confidential
SSA_971281	Accepted		SIL 3	Source and destination address check on packet reception. If the there is a mismatch, the packets are dropped.		N/A	Test	Intel Confidential

3.4.2.2.7 PMC communication over IPC register space

The details of the messaging interface for the SCI and PMC communication over IPC register space are as follows:

1. ISI sends IPC to PMC for Platform Transition Information query:

- ISI will send this IPC to PMC once the runtime FW is downloaded but before starting any Periodic FSTs. This IPC is a query from ISI to know what caused the reboot.

2. PMC sends IPC to ISI for Platform Transition Information response:

- In response to the Platform Transition Info Query from the ISI the PMC will send the following IPC data back to ISI:
    - [0]: Last event G3 exit (cold boot).
    - [1]: Last event type 7 or 8 Global Reset
    - [2]: Last event cause - other. (gets updated by the FW when host_prim_reset is asserted)
    - [30-3] Reserved always reads 0
    - [31] ISI_128_DSW_VALID register bit.
- 3. ISI sends IPC to PMC to Read 128Byte buffer contents:
- ISI sends the offset into the 128B buffer and the size of the data to be read from the DSW buffer as parameters for the IPC request.

4. PMC sends IPC to ISI to Read 128Byte buffer contents

In response to the previous read request from ISI, PMC FW reads the contents of the DSW buffer starting from the offset mentioned in the previous command till the size provided and sends that data back to ISI as a response.

3.4.2.2.8 PMC communication over IOSF-SB

ISI sends IOSF-SB MEMWR packet to PMC to write to 128Byte buffer ISI FW sends a SB MEMWR packet to PMC to write some data to the DSW buffer space. As a part of this SB packet, ISI sends the offset, size and the buffer contents in the IPC message.

3.5 SCI Boot ROM

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590371	Accepted		SIL 3	Following diagram shows the ROM Boot flow in the SCI. Once the SCI ROM pulls in the Runtime firmware from the CSE, it jumps into its entry point to continue the rest of the boot. The detailed SCI ROM flows are shown in the flow diagram below. But at a high level, here's a description of the SCI Boot from the ROM perspective: 1. After PMC patch has brought SCI out of reset and the SCI Has successfully run the POST test, SCI CPU is brought out of reset. 2. The ROM execution starts from the reset vector which is mapped to address 0. 3. ROM disables the interrupts. The interrupts are kept disabled throughout the ROM execution and instead takes the polling approach. 4. ROM sets up the stack and copies the various data sections of ROM to the SRAM. 5. ROM reads local control		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				registers for the fuse bits information 6. ROM then initializes various drivers that are required for its uses. 7. ROM sets up the IPC communication channel with CSE after establishing the IPC Handshake. 8. CSE sends a message to SCI ROM asking it if it needs the Runtime FW. 9. SCI ROM acknowledges this message 10. CSE loads the SCI FW from its NVM and does the verification. 11. Due to the limited availability of CSE SRAM, SCI FW will be divided into chunks 12. For each chunk, CSE sends an IPC message with the source address and the size of the chunk to SCI ROM. 13. The first chunk has the header of the FW image (as described in FW layout and Stitching section) 14. SCI ROM will program the SRAM CRC register sets in such a way that enables the CRC calculation of only the FW and not the associated header. 15. SCI ROM will then program its local DMA to download the FW to the SRAM. 16. Steps 17 to 20 will continue till entire FW is downloaded. 17. During process, if an IPC from CSE contains Error message, SCI ROM will set the Error status in 'scratchpad_m3' register in Msg Box space so that BIOS/OS can query it during Host boot and based on type of error, it can choose to enter the FW upgrade mode. 18. Once the entire FW download completes successfully, ROM will end the CRC calculation by programming the associated SRAM register. 19. It will then compare the computed CRC with the pre-computed CRC present the FW header downloaded from CSE. 20. If there is a mismatch, ROM will drive the NOK and will enter a dead loop.				

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				21. If the CRC matches, ROM will go ahead and interact with CSE for SCI Config data. 22. If SCI Config data is not provisioned by the OEM, CSE will inform SCI ROM about it by setting the appropriate status in IPC msgs. 23. If the Config data is provisioned, SCI ROM will repeat steps 13 to 21 and if the CRC32 check for Config data also passes, ROM prepares to jump to runtime FW. 24. ROM update 'BTOK' status in 'scratchpad_m3' register, sets up rom_context structure, puts its address in R1 register and jumps to SCI FW entry point at address 0x10020. 25. From this point onwards, runtime FW starts executing and that flow is defined in the next section.				

Figure 27 SCI Boot ROM High level flow

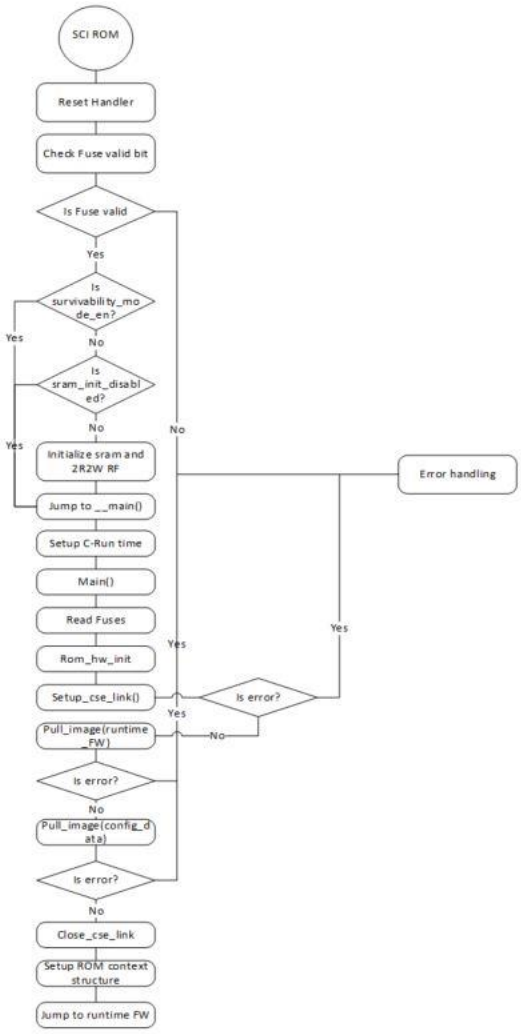


Figure 28 BootROM Flow – CSE IPC Handshake

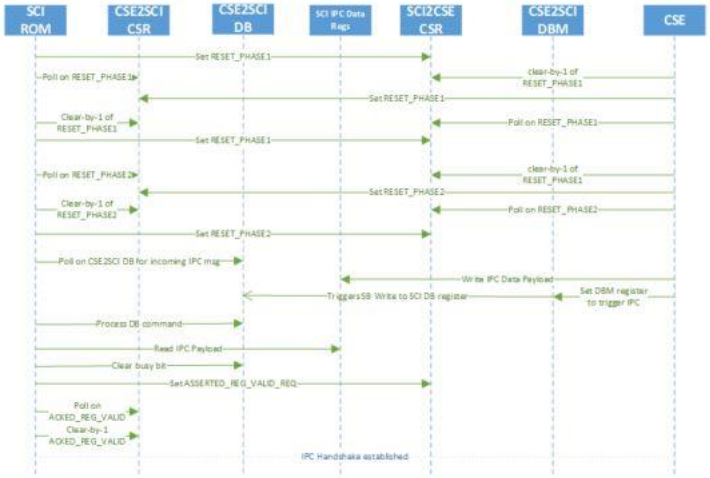
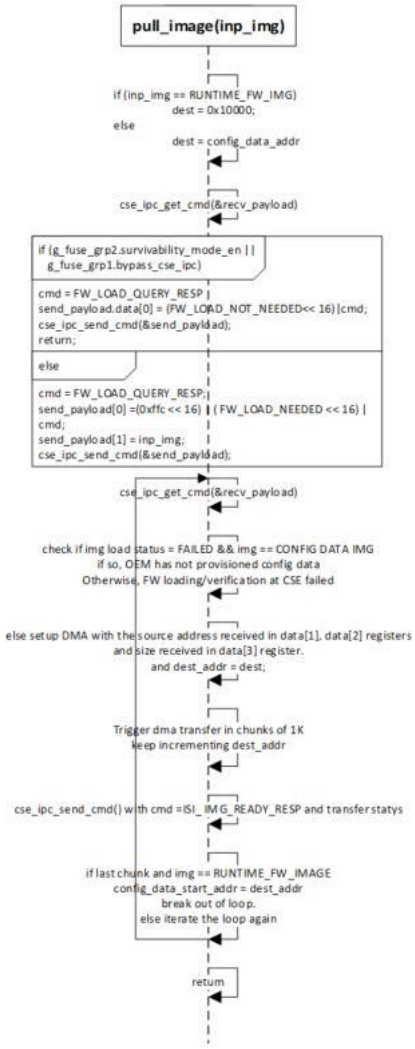


Figure 29 Boot ROM flow - Pull Image



3.5.1 ECC generation for Boot ROM

SCI bootROM will be integrated into the RTL and hence it requires manual generation of ECC using an external tool. The ECC generation tool will generate the ECC for bootROM. The ECC generation tool will take BootROM binary in Intel Hex format as input and generate ECC in Intel HexFormat. ECC generation tool will also prepend the generated ECC to the ROM binary in HexFormat . The command line format for ECC generation tool as follows,

ECCCompute.py -i <inputfile> -o <outputfile> -t rom [-e <address>]

OR

ECCCompute.py --ifile <inputfile> --ofile <outputfile> --type rom [--Error <address>]

inputfile: The input ROM binary file in Intel Hex format

outputfile: The ECC generated output file name in Intel Hex and ARM M3 format

address: The Address in ROM space where single bit and double bit ECC errors are injected. The error injection is optional parameter. For SCI Boot ROM, this address is passed as 0xff00.

3.5.1.1 High Level ECC Generation Tool Flow

The below flow chart explains, the high level flow of the tool. The tool will take BootROM file in Intex HexFormat as input and generate another file in Intel HexFormat as output. The file generated by this tool will contain ECC code for the BootROM. The tool will also prepend the generated ECC to the BootROM Hex file.

Figure 30 Flow Chart for ECC algorithm tool

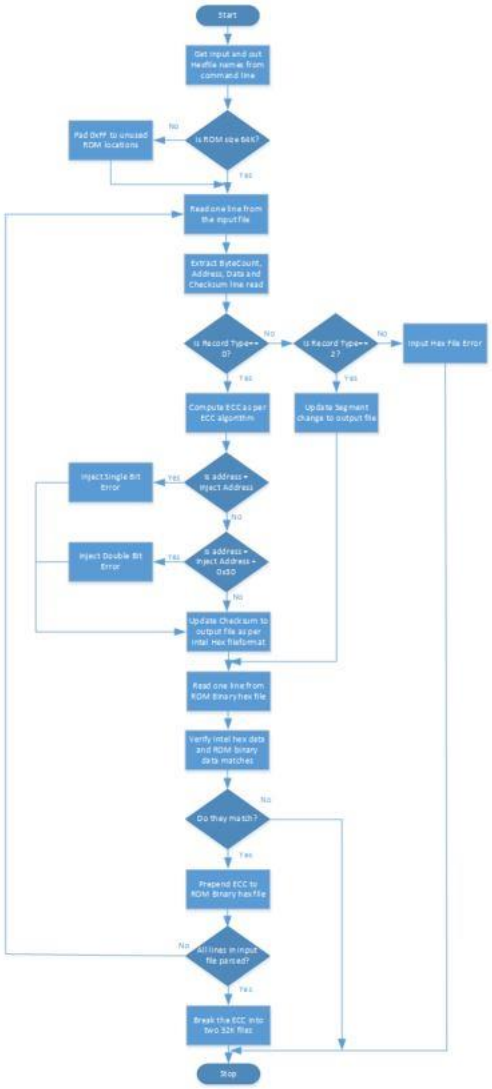


Figure 31 Flow Chart for ECC algorithm tool

3.5.1.2 ECC Algorithm

SCI has 576KB of addressable data and it requires 20 address bits. SCI BootROM has DWORD addressing mechanism and hence it is sufficient to use 18 address lines to access the BootROM. It is a FuSa requirement to compute ECC for both address and Data, the tool will concatenate the address and data and then apply the ECC algorithm. The concatenated address and data (called as Data Wire), will act as input to ECC generation algorithm will look as below.

Figure 32 Data Wire Format for ECC Algorithm

gen_fmi_ecc_fn[0] = (data_wire[2] ^ data_wire[6] ^ data_wire[7] ^ data_wire[8] ^ data_wire[10] ^ data_wire[12] ^ data_wire[15] ^ data_wire[21] ^ data_wire[22] ^ data_wire[23] ^ data_wire[26] ^ data_wire[27] ^ data_wire[28] ^ data_wire[36] ^ data_wire[37] ^ data_wire[39] ^ data_wire[42] ^ data_wire[43] ^ data_wire[45] ^ data_wire[49] ^ data_wire[51] ^ data_wire[55] ^ data_wire[58] ^ data_wire[60] ^ data_wire[61] ^ data_wire[63]);

gen_fmi_ecc_fn[1] = (data_wire[1] ^ data_wire[3] ^ data_wire[6] ^ data_wire[9] ^ data_wire[13] ^ data_wire[15] ^ data_wire[18] ^ data_wire[22] ^ data_wire[23] ^ data_wire[24] ^ data_wire[25] ^ data_wire[29] ^ data_wire[31] ^ data_wire[36] ^ data_wire[38] ^ data_wire[39] ^ data_wire[41] ^ data_wire[43] ^ data_wire[46] ^ data_wire[47] ^ data_wire[51] ^ data_wire[53] ^ data_wire[55] ^ data_wire[57] ^ data_wire[60] ^ data_wire[62]);

gen_fmi_ecc_fn[2] = (data_wire[0] ^ data_wire[3] ^ data_wire[4] ^ data_wire[10] ^ data_wire[13] ^ data_wire[14] ^ data_wire[15] ^ data_wire[16] ^ data_wire[17] ^ data_wire[23] ^ data_wire[26] ^ data_wire[27] ^ data_wire[29] ^ data_wire[32] ^ data_wire[34] ^ data_wire[40] ^ data_wire[42] ^ data_wire[43] ^ data_wire[46] ^ data_wire[48] ^ data_wire[51] ^ data_wire[54] ^ data_wire[58] ^ data_wire[59] ^ data_wire[60] ^ data_wire[62]);

gen_fmi_ecc_fn[3] = (data_wire[0] ^ data_wire[3] ^ data_wire[5] ^ data_wire[9] ^ data_wire[12] ^ data_wire[15] ^ data_wire[16] ^ data_wire[20] ^ data_wire[23] ^ data_wire[25] ^ data_wire[27] ^ data_wire[28] ^ data_wire[30] ^ data_wire[32] ^ data_wire[33] ^ data_wire[35] ^ data_wire[36] ^ data_wire[41] ^ data_wire[43] ^ data_wire[45] ^ data_wire[48] ^ data_wire[50] ^ data_wire[51] ^ data_wire[56] ^ data_wire[57] ^ data_wire[61]);

gen_fmi_ecc_fn[4] = (data_wire[0] ^ data_wire[2] ^ data_wire[4] ^ data_wire[7] ^ data_wire[10] ^ data_wire[14] ^ data_wire[15] ^ data_wire[18] ^ data_wire[19] ^ data_wire[20] ^ data_wire[29] ^ data_wire[30] ^ data_wire[31] ^ data_wire[34] ^ data_wire[35] ^ data_wire[37] ^ data_wire[44] ^ data_wire[45] ^ data_wire[47] ^ data_wire[50] ^ data_wire[52] ^ data_wire[53] ^ data_wire[55] ^ data_wire[57] ^ data_wire[59] ^ data_wire[63]);

gen_fmi_ecc_fn[5] = (data_wire[1] ^ data_wire[5] ^ data_wire[7] ^ data_wire[9] ^ data_wire[11] ^ data_wire[14] ^ data_wire[16] ^ data_wire[17] ^ data_wire[21] ^ data_wire[23] ^ data_wire[26] ^ data_wire[30] ^ data_wire[31] ^ data_wire[33] ^ data_wire[35] ^ data_wire[38] ^ data_wire[39] ^ data_wire[44] ^ data_wire[46] ^ data_wire[47] ^ data_wire[49] ^ data_wire[52] ^ data_wire[54] ^ data_wire[59] ^ data_wire[61] ^ data_wire[63]);

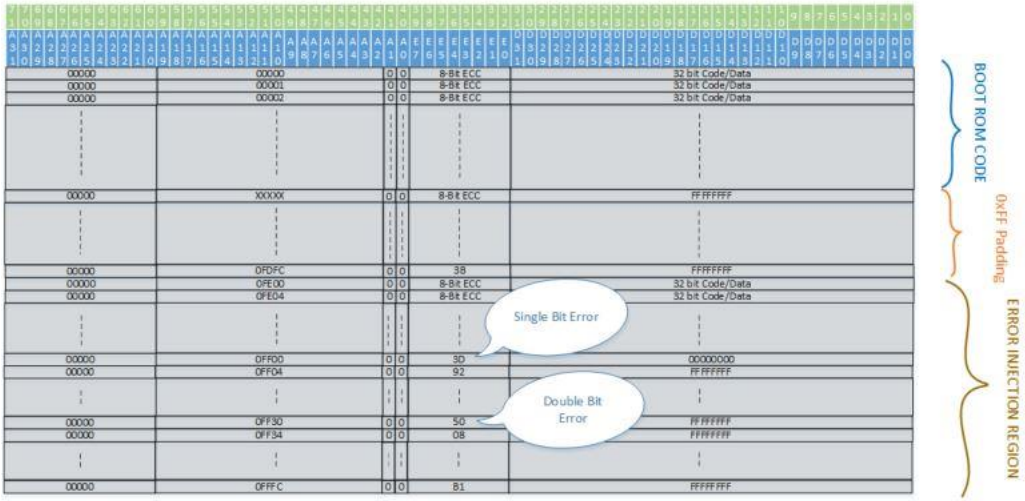
gen_fmi_ecc_fn[6] = (data_wire[2] ^ data_wire[5] ^ data_wire[6] ^ data_wire[7] ^ data_wire[8] ^ data_wire[11] ^ data_wire[12] ^ data_wire[18] ^ data_wire[19] ^ data_wire[21] ^ data_wire[24] ^ data_wire[25] ^ data_wire[31] ^ data_wire[32] ^ data_wire[34] ^ data_wire[35] ^ data_wire[38] ^ data_wire[40] ^ data_wire[42] ^ data_wire[50] ^ data_wire[51] ^ data_wire[52] ^ data_wire[54] ^ data_wire[56] ^ data_wire[59] ^ data_wire[62]);

gen_fmi_ecc_fn[7] = (data_wire[1] ^ data_wire[4] ^ data_wire[7] ^ data_wire[8] ^ data_wire[11] ^ data_wire[13] ^ data_wire[17] ^ data_wire[19] ^ data_wire[20] ^ data_wire[22] ^ data_wire[24] ^ data_wire[28] ^ data_wire[31] ^ data_wire[33] ^ data_wire[35] ^ data_wire[37] ^ data_wire[40] ^ data_wire[41] ^ data_wire[43] ^ data_wire[44] ^ data_wire[48] ^ data_wire[49] ^ data_wire[53] ^ data_wire[56] ^ data_wire[58] ^ data_wire[59]);

ECC = { gen_fmi_ecc_fn[7], gen_fmi_ecc_fn[6], gen_fmi_ecc_fn[5], gen_fmi_ecc_fn[4], gen_fmi_ecc_fn[3], gen_fmi_ecc_fn[2], gen_fmi_ecc_fn[1], gen_fmi_ecc_fn[0] }

3.5.1.3 SCI BootROM organization

Figure 33 Internal representation of ECC in BootRom



The above diagram explains how, SCI 32-bit code/data and ECC are stored in SCI BootROM. During memory read, Code/Data along with ECC will be read by processor. The ECC comparator will regenerate the ECC for Address and Code/Data and compare it with the ECC present in the BootROM. If there is a mismatch then SCI hardware will generate an interrupt.

ECC generation tool will inject single bit error at 0xFF00 address and double bit error at 0xFF30 address. These error injections are used to verify the functionality of ECC engine during SCI boot every time.

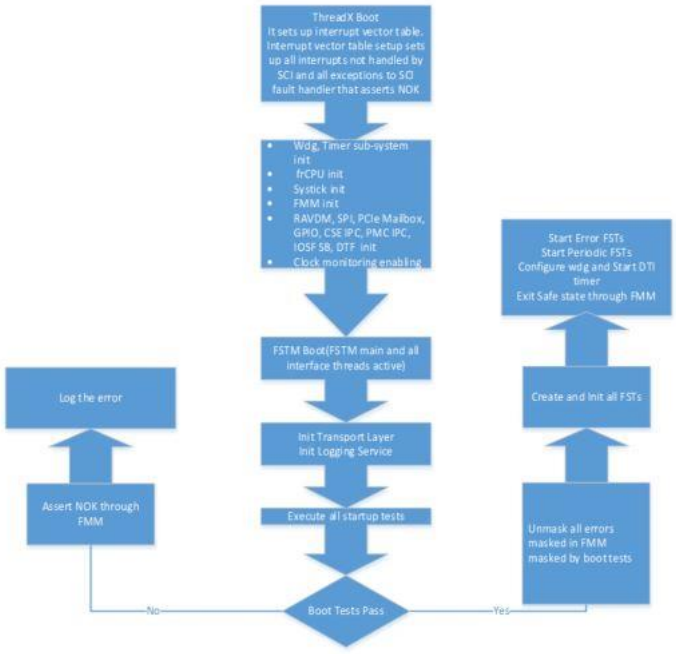
Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 849248	Accepted		SIL 3	0xFF padding in the memories will allow invalid opcode exception if instruction pointer reaches there.		N/A	Test	Intel Confidential

3.6 Firmware Boot Flow

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 590398	Accepted		SIL 3	ROM code jumps to SCI firmware entry point after downloading the firmware image to SCI RAM. Boot process of SCI firmware follows following steps. • RTOS(ThreadX) Boot - ThreadX		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				Boot. It sets up interrupt vector table. All interrupts not handled by SCI and all exceptions are mapped to SCI fault handler that asserts NOK • Initializes the hardware IPs to be used by firmware • Enable Clock Monitoring • FSTM Boot. FSTM main thread, fRCPU Thread and interface threads are spawned. • fRCPU Thread reads NVIC and MPU registers periodically. • Initializes Transport layer, timer service, logging service. • FSTM runs startup tests. • On successful completion of startup tests, FSTM main thread creates and initializes all other FSTs, DTI timer is started and safe state is exited. Any error encountered in any of the above steps causes NOK to be raised even before exiting safe state.				

Figure 34 High-level SCI Boot flow



Boot of SCI after power on till entry point of firmware is mentioned in ROM boot flow.

In addition to the boot flow, the OKNOK pins also go through various states with various platform state transitions. Following table captures these OKNK States with respect to boot/runtime phases:

Table 8 OK/NOK State transitions

Platform/SCI State	OKNOK State
Default state on platform power on	0/0
SCI IP out of reset	1/1
SCI FW Boot Tests and Host Boot STL completed successfully	1/0
SCI FW/HW detected Error	0/1
Elkhart Lake host requested NOK	0/1

3.7 Functional Safety Test Manager (FSTM)

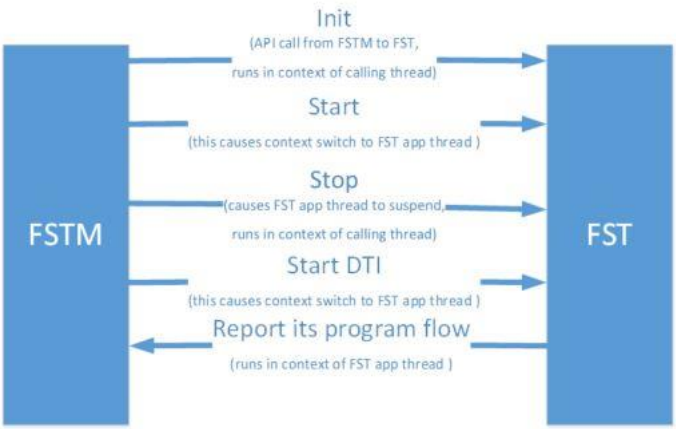
FSTM main thread is the first thread in SCI firmware that ThreadX schedules after firmware boot.

3.7.1 FSTM – FST Interactions

FSTM is responsible for creating various communication interface threads and the FSTs. Internally, each of these steps involve interactions between FSTM and FSTs at various stages and in various scenarios.

FSTM is responsible for creating and orchestrating various FSTs. Below diagram shows at a very high level, the interaction between FSTM and any FST in general

Figure 35 FSTM-FST Interface



Following table describes each of the APIs briefly. We will create a separate document describing the APIs

Table 9 FSTM-FST Interface API's

Interface	Description
Init	Initialize and pass configuration
Start	Activate FST

Stop	Deactivate FST
Start DTI	DTI Window start indication - only for Periodic FSTs

Following sequence diagrams represent these interactions. Each flow diagram details the various command and data exchanges between the FSTM and FSTs and how the infrastructure is designed to handle such interactions

Figure 36 FSTM_FST Create/Init/Start flow

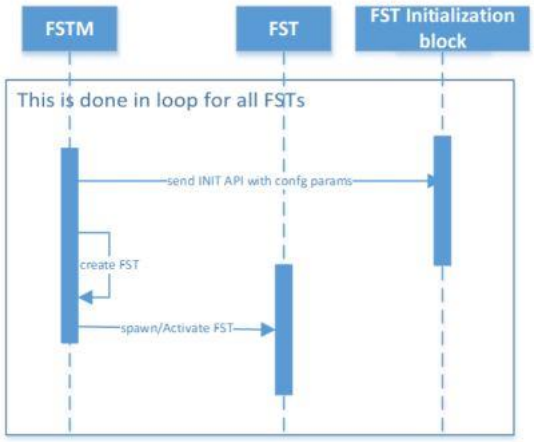
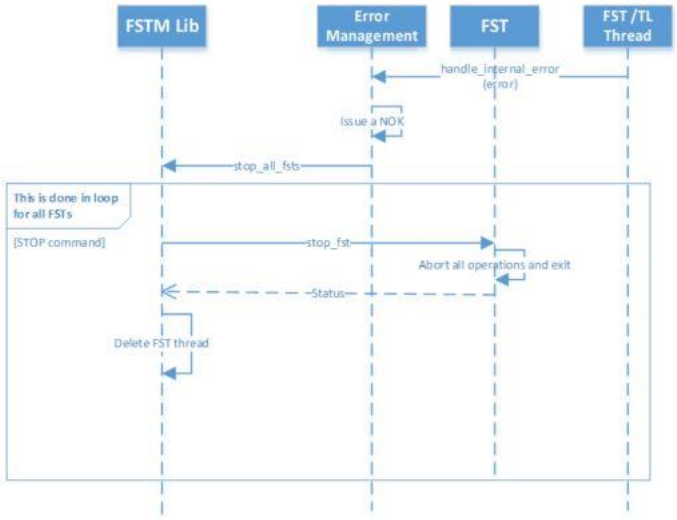


Figure 37 FSTM_FST Stop flow



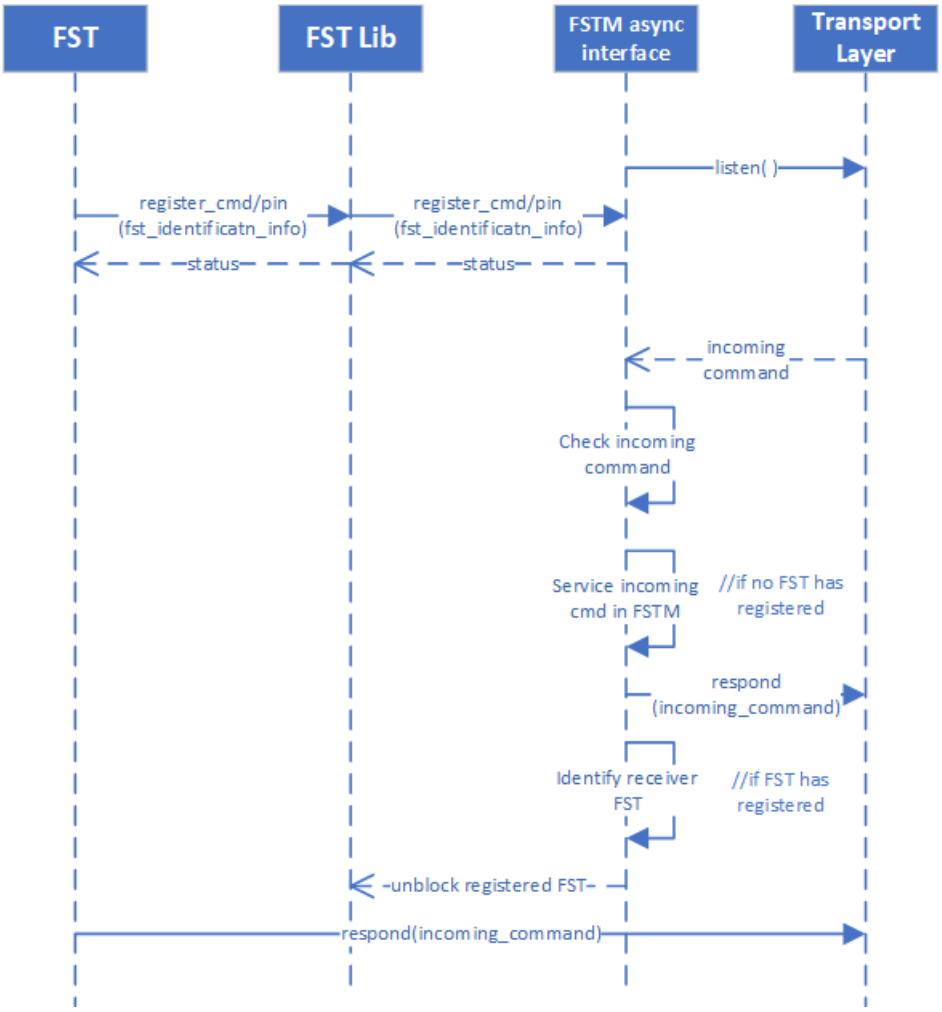
3.7.1.1 Managing Communication with asynchronous communication Interfaces

Asynchronous commands that originate from interfaces that SCI has with external world (peer endpoints connected to SCI through various communication interfaces) are routed to required FSTs through asynchronous interface threads. Some of these commands that need minimal processing are serviced directly by this thread itself. Interface thread parses the incoming command to know if it is to be routed to FSTs or to be serviced by itself. The FSTs that need to

service any particular command can register themselves with the interface threads to forward specific command to them. A FST or service can have registration for multiple commands.

Below diagram, in general, shows the FW infrastructure model designed to handle communication with such interfaces that send asynchronous commands to SCI:

Figure 38 FSTM_FST Communication interface data receive flow



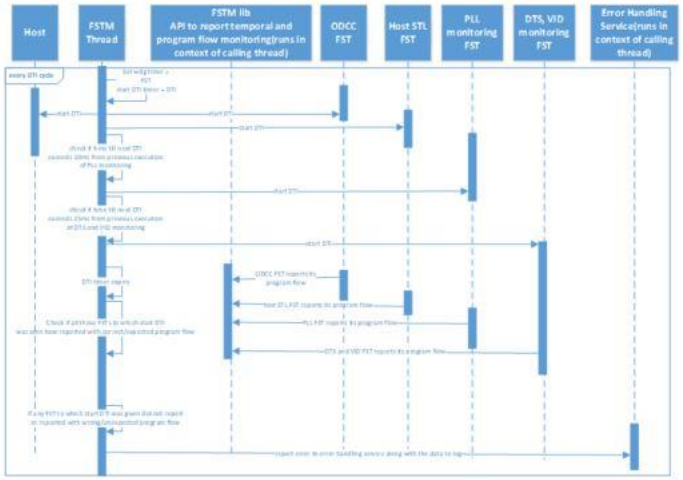
FSTM asynchronous thread is the receiver of commands from any peer endpoint if the command is initiated from them. When any such event happens, FSTM async interface will get triggered so that right FST can be involved to handle the message.

3.7.1.2 Temporal and Program Flow Monitoring

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590419	Accepted		SIL 3	1. FSTM Main Thread configures wdg with PST time, starts DTI timer with periodicity equal to DTB - _STL_ALLOCATED_TIME – T_{ISI_MAX} and informs Host SW and each		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				periodic FST about the start of their periodic cycle for that FST by sending START DTI message to it. Host SW can use this periodic START_DTI message as ISI heart beat 2. FST that is idle is blocked on START DTI message and gets unblocked and starts performing its operations. 3. After performing required operations, it reports its program flow to FSTM through an API provided by FSTM. This API gets executed in the context of calling FST. 4. If FST experiences any error, it either reports wrong value in its program flow or does not report anything at all. 5. On program flow timer expiry, FSTM main thread that is blocked on it gets scheduled. 6. FSTM thread checks if all FSTs that were sent START DTI message from FSTM main thread and were expected to report have have reported and with correct program flow value. 7. In case any of them did not report within their periodic cycle or reported with wrong value, NOK is asserted by FSTM 8. If there's no error, FSTM will reload the WDT 9. All steps above get executed every program flow interval on program flow timer expiry. The program flow timer interval is the lowest time cycle value that can be used to regulate the periodicity of all the FSTs executing in the firmware. So through above steps, any error that FST encounters gets reported to FSTM, e.g. if PLL monitoring FST does not get any response to command it sent or any FST does not receive its expected command from Host, it will report its program flow to FSTM. On DTI timer expiry, FSTM main thread is able to detect it and will assert a NOK. Similarly if PLL out of lock or ODCC comparison failure happens, this also gets reported by FST to FSTM. On DTI timer expiry, FSTM detects unexpected program flow data from the FST. In both above conditions, FSTM causes NOK to be asserted.				

Figure 39 Temporal and Program Flow Monitoring



3.7.1.3 fRCPU Initialization

Before FSTM can go ahead and initialize any interfaces or run any of its FSTs, it must initialize the fRCPU.

As per fRCPU_armcm3_A1.1.3_deployment_on_SCI.docx, SCI has 104 msec from its boot time to mask the fRCPU timeout error. Failing to do so would result in NOK assertion by FMM.

In order to configure the fRCPU, the FW must perform the following functions:

1. Enable fRDI by setting Debug Exception and Monitor Control Register to value 0x01000000 (Bit24 = 1)
2. Mask fRCPU_armcm3 Timeout Error before 104 msec from Reset
3. Set the fRCPU_PST Time
4. SW shall mask the fRCPU outer errors “PSTErrorOut”, “RWErrorOut” and PARErrorOut.
5. Enable fRCPU_armcm3 CORE.
6. Execute 5 consecutive NOP instructions to synchronize ARM CM3 CPU and fRCPU.
7. Perform GP Register initialization- (R0-R14 including to two SPs)
8. When fRCPU_armcm3 is enabled, until the end of the startup procedure, the ARM Cortex-M3 state must be as when out of reset:
    - The processor is in Thread mode, priority is privileged, and the stack is set to Main
    - All the exceptions and interrupts should be NOT ACTIVE and NOT PENDING
    - SysTick should be not initialized or fully re-initialized after this procedure, before any of its features (such as SysTick interrupt, COUNTFLAG event etc.) is used
9. MPU initialization
10. NVIC register initialization
11. After the above have been performed, unmask the CPU outer errors “PSTErrorOut”, “RWErrorOut” and PARErrorOut, which were previously masked

3.8 Functional Safety Tests (FST)

SCI is responsible for running or orchestrating various functional safety tests on itself and host CPU. The FW SCI component responsible for this is called as the Functional Safety Test (FST). We divide these tests into following categories:

Table 10 FST Types

#	FST	Description
1	Periodic FST	SCI FST is responsible for ensuring the diagnostic has happened within DTI and results are available. FSTM managed timer to ensure results come within DTI.
2a	On demand signal/event driven FSTs	Error handling, any other interrupt driven activities.
2b	Special purpose one time FSTs	Boot FST for performing boot time activities.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590427	Accepted		SIL 3	Category 1 are timer driven and have deadlines associated to meet FuSA requirements. These tests need to be completed in each of DTI window. Category 2a and 2b on the other hand are event driven. They are run when a SMCLparticular event happens on the platform e.g. error interrupts or on boot. FSTM manages these tests. FSTM is the main application that is the first one to be launched by the RTOS. There is only one FSTM in the SCI FW infrastructure. It is also responsible for managing the timer associated with platform DTI and EHL Reaction Time budget (PST = DTI + T_{ISI_MAX} + T_{RES_MAX}). Platform PST timer is maintained through watchdog. In case of undetected. PST violation by firmware, watchdog will fire and cause NOK. If FSTs need additional timers, they are left to FSTs to manage. FSTM is responsible for the following: 1. Read and configure various FSTs 2. Launch and manage various FSTs 3. Scheduling various FSTs based on the configured policy 4. Read configuration data, configure PST/DTI and manage platform PST/DTI timer and send messages to all periodic FSTs and also the host upon start of this timer. Each functional safety diagnostic application is built into a FST. Each FST is a RTOS thread which may launch separate sub-threads based on its functionality. In later releases of ISI, we may use ThreadX Modules instead of just threads for each		N/A	Test	Intel Confidential

				FST.				

3.8.1 Type 1: Periodic FSTs

These **category 1** FSTs primarily are responsible for monitoring of safety operations/tests for various components of the platform. One good example is the STLs running to verify the ODCC functionality implemented in host SW in Elkhart Lake.

SCI responsibility here is to ensure all the required tests are run and results are made available in a safe and trust worthy fashion. SCI will process the results and take appropriate action if any failure is detected. SCI will also ensure these tests are repeated every PST.

The overall flow is as follows:

1. On Init
    - Get configuration from FSTM and process it as needed
2. On Start
    - FST thread is spawned
3. On StartDTI
    - Start the required FST processing. This will include either sending commands to external entities and expecting responses coming asynchronously. Some FSTs do not wait for this message to start FST processing and are always running based on asynchronous inputs coming from host e.g. ODCC FST. On getting StartDTI, such FSTs will just indicate to supervisory thread (FSTM main thread) that it is alive and running
    - As the responses arrive, process them.
    - If all processing is complete, inform the FSTM watchdog which is doing program flow monitoring

3.8.2 Type 2a: On Demand Signal/Event Driven FST

These FSTs are executed multiple times as per the signal or event. FSTs handling various errors happening on the Compute or PCH side reaching SCI through various GPIOs fall in this category.

3.8.3 Type 2b: Special purpose one time FST

These category 2a FSTs are responsible for performing some special purpose and primarily one time safety operations. For Elkhart Lake platform, these include boot time testing. The trigger to execute these FSTs will come either from firmware boot or host (e.g. message from BIOS to indicate Host has booted).

SCI responsibility in this case is to orchestrate these tests and ensure the results are satisfactory for the safety of the platform. SCI will drive the OK-NOK pins to drive the system into Safe State.

The overall flow is as follows:

1. FSTM initializes the FST at SCI boot time.
2. Execute the FST when the trigger comes from FSTM.
3. If there is an error during FST execution, SCI software drive the NOK pins and drive the platform to safe state.

3.9 Freedom from Interference (FFI)

There are following primary reasons that requires a software to have mechanisms for freedom from interference.

1. Mixing of different criticality components
2. Implementation of diverse and redundant software
3. One software component acting as safety mechanism for another software component

In the first case, if there are components at mixed integrity level (ASIL or SIL levels) co-hosting on SCI, then we need to ensure there is clear freedom from interference. This means clear spatial and temporal separation using HW protection, ensure interfaces are cleanly defined to ensure minimal interference etc. For SCI, our POR includes the following:

- All code at same criticality – ASIL D or SC-3
- All code loaded in SRAM
- No runtime loading of FW.
- No Intel or customer secrets in the FW

For the second and third, SCI firmware is not implementing any diverse and redundant software components for same functionality nor it is implementing any safety software component that acts as safety mechanism for another software component within SCI firmware.

Hence there is no interference source within the SCI firmware.

Debugs are added in debug build and not in release builds. SCI additionally supports Intel MIPI SysT traces supported but they are turned off field using the configuration data.

3.10 Software Development Kit

3.10.1 Tools support

To help Intel developers and OEMs, we will need to enable following set of tools.

1. To integrate various components of firmware in a given flash layout and as per CSE requirements. This tool will help generation of firmware, configuration parameters as per CSE requirement. This tool is called as Firmware Build Integration Tool.
2. To configure various parameters for firmware. This tool is called as Configuration Management Tool

All these tools need to be safety compliant as per IEC 61508 standard.

3.10.1.1 Firmware Build Integration Tool

The SCI Build integration tool will perform following actions:

- Build SCI bootROM
- Build SCI firmware
- Generate ECC for SCI bootROM
- Inject ECC errors for bootROM
- Generate CRC for SCI firmware

- Sign SCI firmware for integrity check
- Generate SCI Configuration parameters

The SCI firmware will be downloaded into SCI SRAM, after integrity check by CSE. CSE has limited SRAM and cannot verify the integrity of SCI firmware in one go. To address this CSE limitation, the Build Integration Tool will break the SCI firmware into smaller chunks and then present it to MEU tool to generate integrity check header. MEU tool will stitch these smaller chunks after prepending the header. Build Integration tool will repeat the same process for Configuration binary and generate a separate binary. The generated SCI firmware sub-partition and Configuration Data sub-partition will be presented to IFWI tool to integrate the SCI images into IFWI image.

After integrity check, BootROM will perform DMA transfer of SCI firmware present in flash. The inbuilt hardware CRC32 engine will compute CRC-32 for transferred firmware. BootROM will compare the CRC32 present in the SCI firmware header with the CRC32 generated by CRC engine and if there is a mismatch the bootROM will halt the boot process. Below is the format of header with CRC32 prepended to SCI firmware.

Table 11 Header Format

Number of Bytes	Field Name	Comments
4 Bytes	CRC-32	32-bit CRC for the configuration Data including the Header
8 Bytes	Signature	The signature will indicate start of configuration data. This will be always $SCIBIN$
1 Bytes	Major Version	This will be the Major version of SCI firmware
1 Bytes	Minor Version	This will be minor version of SCI firmware
18 Bytes	Reserved	Reserved for future use

3.10.1.2 Configuration Tool

The Sycamore Island (SCI) Configuration Tool will contain command line interface. GUI interface for SCI configuration tool is not a requirement and will not be supported. The configuration data will be a static data and needs to be created using configuration tool during firmware build time. As per current architecture it is not possible to update the configuration data during runtime. Once Elkhart lake board leaves factory floor, any change to configuration data requires SCI firmware update.

3.10.1.2.1 Configuration Tool Command Line Interface

The command line interface will contain three files.

1. A script/binary which will read configuration data from .ini file and generate 64K configuration data as mentioned in this document
2. Ini file, which will define keywords and their parameters. This input will define the properties of the keyword and will be used to validate the configuration parameters. Let us call this ini file as keywords.ini

3. Ini file, which will contain all configuration parameters and will act as input to configuration tool. Let us call this ini file as SCIConfig.ini

Keywords.ini format

[Keyword]
Description = Description about the configuration parameter in text

Type = Integer | String | Boolean | Float
MinimumRange = Minimum supported value
MaximumRange = Maximum supported value

Scope = FST / Global
Supported Values = Value1, Value2....

[Global]
Keyword = Value
Keyword = Value

[FSTName]
Keyword = Value
Keyword = Value

3.11 Configuration Management

FSTM will parse through the configurations data stored in dedicated SCI config area in RAM. In this dedicated SCI config area, there are configurations to be consumed by SCI firmware and by host software stack. For the configurations to be consumed by FSTs in SCI firmware, FSTM will pass these configurations to the FSTs through the FSTs init functions. For the configurations to be consumed by host software stack, consumer of configuration from FSTM is Startup tests. Startup tests provides it to host as part of handshake at the start up. The configurations are also returned to different workloads when they register themselves with SCI.

3.11.1 Sycamore Island Configuration Format

64 bytes from the base address of configuration space are reserved for Configuration Header, and then the configuration parameters of every FST follows the header. Below table explains the configuration format

Table 12 Configuration Format

Number of Bytes	Field Name	Comments
4 Bytes	CRC-32	32-bit CRC for the configuration Data including the Header
8 Bytes	Signature	The signature will indicate start of configuration data. This will be always $CONFIG$
1 Bytes	Major Version	This will be the Major version of SCI firmware
1 Bytes	Minor Version	This will be minor version of SCI firmware
18 Bytes	Reserved	Reserved for future use
1 Bytes	No of FSTs	Number of FSTs excluding the global configuration

Number of Bytes	Field Name	Comments
2 Bytes	Total Size	Total size of configuration Data
29 Bytes	Reserved	Reserved for future use
Variable	FST-1 Config Data	Configuration parameters for FST-1
Variable	FST-2 Config Data	Configuration parameters for FST-2
…….	…….	…….
…….	…….	…….
Variable	FST-n Config Data	Configuration parameters for FST-n

3.11.2 FST Configuration

SCI firmware allows individual FSTs to be configured as per customer requirements. The FST configuration contains 32 bytes configuration header followed by configuration data. Each individual data is delimited by { } and SCI firmware should look for these delimiters to differentiate each configuration parameter. The format of FST configuration as follows.

Table 13 FST Configuration Format

Number of Bytes	Field Name	Comments
8 Bytes	Signature	The signature will indicate start of FST configuration data. This will be always $FSTCFG$ for every FST
16 Bytes	FST Name	The name of FST
1 Bytes	FST ID	The ID of the FST. Used to map FST config data to actual FST in the firmware
1 Bytes	No of Entries	Number of FST configuration parameters present in the FST
2 Bytes	FST Size	Size of FST
4 Bytes	Reserved	Reserved for future use
Variable	Config Param 1	FST configuration Parameter 1
Variable	Config Param 2	FST configuration Parameter 2
…….	…….	…….

4.0 Elkhart Lake Platform FSTs

This section will detail various FSTs planned for Elkhart Lake platform. These are:

- Category 1: Periodic FSTs
  - Host STL interactions FST
  - PUnit associated FSTs
    - PLL lock monitoring FST
    - Voltage and Thermal monitoring FST
  - SCI STL FST
  - ODCC FST
- Category 2: On Demand Signal Driven FSTs
  - Corrected Error Handler FST (ERR0)
  - Non Fatal Error Handler (aka ERR1)
  - FATAL Error Handler FST (ERR2)
- Category 2b: Special purpose one time FSTs
  - Boot FST

4.1 Host STL Interaction FST

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590483	Accepted		SIL 3	This FST is responsible for • Ensuring that the right number of STL results come from the Host SW periodically and verify all the results and generate NOK if needed • Ensuring that consecutive STL result from each STL manager is not more than configured time apart Host will then run the STLs on all cores and send all the results in one or more messages to SCI. The FST will update OK/NOK status as per the results. If any one or more of the safety cores assigned to the SM do not log the results in predetermined		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				time, FST will indicate NOK. In the configure command, the STL manager will send the additional parameters to let SCI know the interval at which STL results would be sent to SCI. Customers can compute these timer values using the below formulas based on their Safety Loop implementation: Maximum STL Interval time (T_{MAX_STL}) = Safety loop time(T_{CYCLE}) + allowed maximum delta + ISI processing time for a message + T_{RES_MAX} Minumim STL Interval Time (T_{MIN_STL}) = Safety loop time(T_{CYCLE}) - allowed maximum delta + ISI processing time for a message(T_{ISI_MAX}) + T_{RES_MAX} $T0_{IDEAL\_STL}$ = Ideal time difference between Safety Loop invoking Start Workload and sending the first STL result + T_{ISI_MAX} + T_{RES_MAX} $T0_{MAX\_STL}$ = Maximum time difference between Safety Loop invoking Start Workload and sending the first STL result + T_{ISI_MAX} + T_{RES_MAX}. ISI will assert a NOK if it doesn't receive first STL Result from host within this time. ISI FW will enforce bounds checking for all these parameters before consuming them. Using these, SCI FW will compute the time period within which it must receive the next STL result from the host SW. STL_Timer in SCI = T_{MAX_STL} + σ (T) – T_{ISI_MAX}, where σ (T) is calculated as σ (T) = Safety Loop Time(T_{CYCLE}) – time difference between current and previous STL result received + σ (T-1). For the first STL result, σ(0) is				

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				computed as: $\sigma(0) = T0_{IDEAL\_STL}$ - (Timer shapshot on receiving the first STL Result – Timer snapshot on receiving Start Workload command) The deviation mentioned above could be negative if results arrived little later than the expected time (but before the maximum tolerance time). SCI FST will ensure STL results time arrival for a given loop is between T_{Max_STL} and T_{MIN_STL}.				
SSA_ 850159	Assumed		SIL 3	**AoU1:** When the Safety Loop periodicity is set to 1 msec the number of messages sent to SCI from host related to STL results is limited to one.		N/A	Test	Intel Confidential
SSA_ 850158	Assumed		SIL 3	**AoU2:** Time between two messages from same workload must be greater than or equal to 1msec		N/A	Test	Intel Confidential

4.1.1 Flow Diagram for Host STL Interaction FST

The flow diagram for the Host STL Interaction FST is as given below. The Host STL Interaction FST receives the host side software configuration parameters during its dispatch from FSTM.

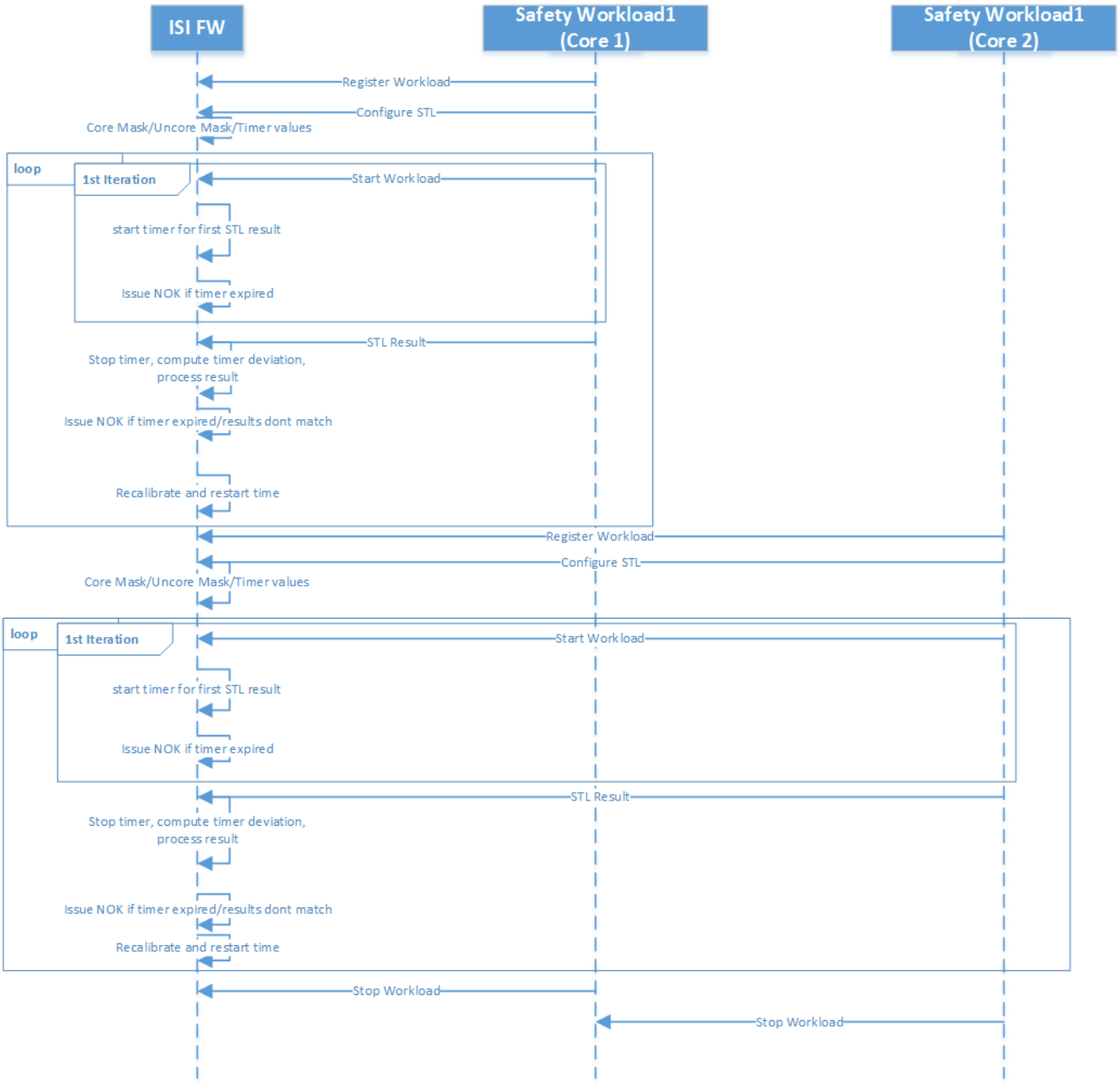


Figure 40 High-level flow diagram for Host STL Interaction FST

To keep the flexibility we will architecturally support presence of multiple STL managers in the system.

DTI timer: This DTI timer has to account for the time that is elapsed between STL mgr generating the STL results message and SCI receiving it.

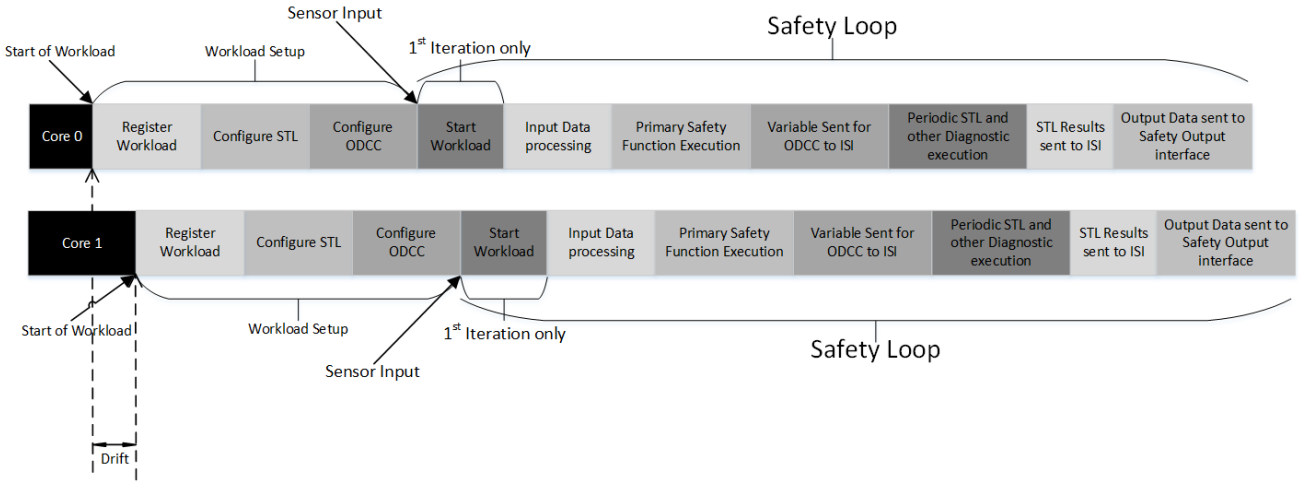
4.1.2 Message formats used by Host STL Interaction FST

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 590488	Accepted		SIL 3	To enable periodic monitoring of HOST STLs, the host SW application needs to perform the following steps in below order: 1. Register workload – Register the safety application with the SCI FW. SCI FW will assign a unique Workload ID to the calling application. The workload can be either a workload that is also executing safety function or an independent workload that executes only STLs 2. Configure STL workload – This command will allow host application to let SCI know what STLs would this application run – Core STL, UnCore STL and OEM STL. In addition it will also tell SCI the following: a. Safety Loop time(T_{CYCLE}) b. Maximum STL Interval Time(T_{MAX_STL}) c. Minimum STL Interval Time(T_{MIN_STL}) d. ($T0_{IDEAL\_STL}$)Ideal time difference between Safety Loop invoking Start Workload and sending the first STL result e. Maximum time difference between Safety Loop invoking Start Workload and sending the first STL result ($T0_{MAX\_STL}$) 3. Start Workload – This command is an indication to SCI that the Safety Loop has started and SCI FW should expect the STL Results (and ODCC snapshots if that is also included in the Safety loop) from the workload from this point onwards. This Command is sent only during the first Safety Loop execution and includes a bitmask of		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				what functions will run as part of Safety loop. Currently only Host STL and ODCC are the only options supported. 4. STL Result Send – This command will allow Workload to send the STL results to SCI. It will include the bitmask of Core STLs run, their status, bit mask of uncore stl run and their status, bitmask of OEM STLs run and their status and various error counts. SCI FW will compare these results and bitmasks with the expected results (calculated based on the details sent during the Configure command) and will assert NOK if any error is found. 5. Stop Workload - This message is sent by safety application when it wants to stop a periodic ISI function (Host STL or ODCC) execution. SCI will no longer expect the corresponding results from the Host. Details of these commands can be found in the ISI Diagnostic Specification.				

4.1.3 Allowed Maximum Delta in Safety Loop

Figure 41 Sample Safety Workload



Above diagram shows a sample Safety Workload. It primary consists of a safety loop, which itself is composed of a call to various sub-modules. In the above example, input data processing, primary Safety Function, ODCC snapshot send, Periodic STL execution, STL Result send and output data generation are the primary sub-modules.

Due to various latencies associated with HW such as Input signal transmission, cache-miss etc.; and with SW such as scheduling latencies etc, each of these sub-modules may suffer from execution time variability. 'Allowed Maximum Delta' parameter used in the equations shown above allow customers to approximate such variabilities in these sub modules' execution and use that in the Timer value calculations that are sent to ISI as monitoring timeout values.

As is evident from the sample workload diagrams, T_{MAX_STL} is dependent on execution time variabilities of various sub-modules that are run before the STL results are sent to ISI for processing. Thus, for Periodic STL's T_{MAX_STL} 'Allowed Maximum Delta' value should include the HW/SW variabilities associated with all of them.

4.2 PUnit associated FSTs

4.2.1 PLL lock Monitoring FST

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590500	Accepted		SIL 3	FSTM main thread that invokes all FSTs based on their periodicity requirements shall invoke this FST based on following: • If the platform PST is less than 10 msec then this FST shall be invoked at periodicity slightly less than or equal to 10ms. • If the platform PST is greater than or equal to 10 msec, this FST shall be invoked at same periodicity as DTI corresponding to platform PST.		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				This is done as PUnit handling of all SCI commands are in a "slow" loop; this means response time from PUnit could be more than one msec. The FST will periodically read the following registers through PUnit. • PLL lock status registers: There are four PLL lock status registers in EHL compute die and these correspond to core PLL, SA PLL, ring PLL and CMI PLL. This FST will read these 4 PLL status every time this FST runs through PUnit RAVDM service to ensure all the PLLs stay locked. When any PLL status register value indicates that PLL is not locked, this FST will assert invoke error handling flow that assert NOK.				

4.2.2 Voltage, and Thermal Monitoring FST

FSTM main thread that invokes all FSTs based on their periodicity requirements shall invoke this FST based on following:

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_971282	Accepted		SIL 3	1. If the platform PST is less than 50 msec then this FST shall be invoked at periodicity slightly less than or equal to 50 ms. 2. If the platform PST is greater than or equal to 50 msec, this FST shall be invoked at same periodicity as DTI corresponding to platform PST.		N/A	Test	Intel Confidential
SSA_971283	Accepted		SIL 3	3. Compute die thermal monitoring: EHL compute die has various DTS for reading the temperature data. There are: 1 DTS for core, 1 DTS for SA, 1 DTS for DE, 1 DTS for Thermal and 9 DTS for graphics engine. This FST periodically reads the temperature values using PUnit RAVDM service and takes action if the temperature trends towards configured PROCHOT (default 110C: _CONFIGURED_TEMP_ALERT_VALUE_) and/or THERMTRIP (135C: _THERMTRIP_VALUE_). This FST alerts the host and supervisory component by issuing an early warning when any of the DTS temperature reaches the PROCHOT threshold and is below THERMTRIP threshold (135C). Also, when the temperature trends towards the THERMTRIP threshold (135C), this FST drives the NOK to put the system in safe state. It also does plausibility check between two consecutive positive DTS values and will drive NOK if the temperature difference is greater than 20C		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				(_TEMP_PLAUSIBILITY_LIMIT).				
SSA_971284	Accepted		SIL 3	4. Voltage domain monitoring (monitors voltage and temperature of compute die): There are 4 voltage domains in EHL compute die – VCCCORE, VCCL2, VCCRING and VCCSA. This FST periodically queries the voltage and temperature corresponding to the working point for each of the voltage domain using PUnit RAVDM service. The voltage and temperature read are then verified against voltage/temp pair computed by PCode algorithm present in the FST. Pcode uses same algorithm for voltage correction with temp variations. If the difference between read voltage/temp pair and the values resulting from Pcode algorithm does not exceed the threshold defined in FST, the test is pass else the FST will invoke error flow that asserts NOK.		N/A	Test	Intel Confidential

4.3 On Demand Cross Comparison (ODCC) FST

This section, instead of focusing only on the FST in SCI FW, also provides the details associated with host side SCI SW stack usage. This will give an end to end system level perspective for the ODCC feature.

The idea behind ODCC is to increase the DC by running the same workload on two different physical cores in EHL (could be in same channel or across two channels) and comparing the core and other logic state every PST. The two copies of the workload will be behaviorally same even if there are differences in code (in most cases, they will be identical). The program logic for both the instances is expected to produce same output along with internal state.

4.3.1 ODCC platform configurations

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590506	Accepted		SIL 3	There are two different platform configurations supported in EHL platforms. They are 1oo1 configuration which is based on a single channel and 1oo2D cross monitoring which uses 2 channels. 1oo1 has two possible configuration 1oo1 basic and 1oo1 smart. An EHL CPU with its SCI is considered as a single channel. In case of 1oo1 platform configuration, the workloads are duplicated across 2 cores of the single EHL CPU (known as ODCC pairs) and the SCI performs the comparison operation.		N/A	Test	Intel Confidential
SSA_971286	Accepted		SIL 3	In case of 1oo2D cross monitoring platform, the workloads are duplicated across 2 cores across the 2 channels and both the SCI performs the cross comparison operation. In this case, the 2 cores across the channels form the ODCC pair. The 1oo2D cross monitoring platform is used for applications requiring higher safety levels such as SIL-3.		N/A	Test	Intel Confidential

4.3.2 Host software stack configurations

From host SW stack perspective there are different ways to achieve this:

- Single OS where system SW ensuring the workload/application runs on two cores in parallel but within certain time drift. In this case combination of application, middleware and OS will ensure same workload or application gets executed in parallel on two cores with in certain time drift. In this case the expectation is that I/O is managed in such a way that the application does not see the difference i.e. both instances get the input at the same time without extra delays (E.g. using master/slave mechanism where one of the workload instance acts as a master and owns the I/O and other instance receives the IO from master OR using any other industry mechanisms based on black channel usage).

- **NOT POR:** Two VMs running on two separate cores run same workload or application in parallel with third VM managing the SCI interaction (in addition to other service related housekeeping). In this case, VMM or other system software mechanisms ensure the applications get started at the same time and similar to first option I/O is managed using black channel or other mechanism to ensure both application instances get the copy of input at the same time (i.e. without extra delay).

The system will support multiple applications / workloads running in parallel (within certain permissible drift) on the same two cores using OS based time slicing mechanism.

To minimize the FFI between the two instances of workload executing on two different cores, separation of memory address spaces is provided by use of VMs or the OS. This assumption is made to ensure FFI between the two instances and ease of use/development/deployment.

AoU3: If the workload is not executing continuously, we are going to claim that SCI ODCC FST will not be active while ODCC workloads are not active. This is because, the required DC for these workloads is needed only when that workload is active. So, we will introduce ODCC start and ODCC stop as two different messages.

AoU4: Any common resources used by workload should be considered as part of black channel so that it does not affect the independence between 2 channels.

4.3.3 PST requirements and its impact on number of workloads

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 849269	Assumed		SIL 3	The maximum number of ODCC workloads and the PST requirements are interrelated. As the PST requirements get tighter, the number of workloads supported by the SCI FW reduces. This is primarily driven by number of snapshots SCI can consume within the allotted PST time. Thus, there are two different approaches taken by the ODCC architecture to support both scenarios of PST. They are as given below, - ODCC for PST requirements around 1 mS (Referred as ODCC Approach 1): This is meant for systems where the PST varies around 1 ms i.e. 1-4 ms. The number of workloads that can be supported in this case is limited i.e. about 1 to 4 workloads at most. - ODCC for PST requirements around 10 mS (Referred as ODCC Approach 2): This is meant for systems where the PST requirement is around 10 ms. The number of		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				workloads that can be supported is up to 10. The fundamental difference between the two approaches is that for PST requirements around 1ms, the snapshots are sent directly to SCI-FW without any accumulation in the host. Also, the snapshot transfer from workload to SCI is driven by the workload.				

This version of SAS covers only the first approach to keep implementation simple. The rest of this document provides further details on this approach. Once the customer requirements are further clarified, second approach flows will be added to the SAS.

Refer FuSa User Guide for configuration parameters details.

We take the following requirements from number of workloads to be supported:

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 849270	Assumed		SIL 3	For PST of 1msec, maximum number of WLs will be 1. For 2msec, it will be 2; for 3 msec, it will be 3.		N/A	Test	Intel Confidential
SSA_ 849271	Assumed		SIL 3	For PST of 4 msec or more, maximum number of WLs will be 4. If more WLs need to be supported for higher PST values (more than 4 msec), we will need to use approach #2.		N/A	Test	Intel Confidential

4.3.4 Time drift across workloads

In all the ODCC configurations/cases (all combinations of host software configuration as well as platform configurations) the workload may run for few milliseconds to few months/years as it is very much customer application/usage specific. Since the OS is running the workload on two cores, it is expected that the two cores will NOT be executing exact same set of instructions at the same time. There will be some time-lag between two core executions.

Given the requirement that a failure in ODCC must be detected before PST, this means that the ODCC workload must send the snapshots every 1 msec (which is also the smallest PST supported). In case of 1oo2D cross monitoring platform, the drift issues becomes more relevant. This is due to the reason that the SCI is required to communicate the collected snapshots across the serial links for the peer SCI to perform the comparison of snapshot. The communication latency over serial links and time taken to perform the comparison reduces the permissible drift for the workloads.

In case of EHL platforms, the serial link used is a pair of SPI links for inter-SCI communication. More details on this interface is provided in Section “SCI-SCI communication” and Section “ODCC end-to-end flows”.

AoU5: It is left up-to customer SW stack implementation to synchronize the start of Safety loop of the two channels participating in the ODCC and that they are within the permissible cross channel drift criteria. There after, SCI FW internally makes sure that it asserts a NOK when any of the channels drifts beyond the allowed limit.

The SCI provides the synchronization of workloads at the start as explained in the section below. For runtime drift management, only mechanism SCI provides is the message for Start DTI every DTI interval. The workloads are expected to manage the drift using this and other OS provided mechanisms.

4.3.5 ODCC end-to-end flows

This section discusses the end-to-end flows for 1oo1 and 1oo2D configurations. The configuration of the platform, whether 1oo1 or 1oo2D configuration, is provided by the host configuration data in the SCI Config Data.

Customer is responsible for computing "Maximum snapshot_interval", "Minimum snapshot interval" and other parameter values, as described below, for a given workload based on customer application requirements and Intel published AOUs as follows:

- *Maximum snapshot_interval (T_{MAX_ODCC})* = T_{CYCLE} + Allowed Maximum Delta + T_{ISI_MAX}
- *Minimum snapshot_interval (T_{MIN_ODCC})* = T_{CYCLE} - Allowed Maximum Delta + T_{ISI_MAX}
- *CROSS_CHANNEL_DRIFT = T_{CYCLE}* – Allowed Maximum Delta – T_{ISI_MAX} – Time taken by other Safety Loop compnents that run before ODCC.
- *$T0_{IDEAL\_ODCC}$* = Ideal time difference between Safety Loop invoking Start Workload and sending the first snapshot for comparison + T_{ISI_MAX} + T_{RES_MAX}
- *$T0_{MAX\_ODCC}$* - Maximum time difference between Safety Loop invoking Start Workload and sending the first snapshot for comparison + T_{ISI_MAX} + T_{RES_MAX}. *ISI will assert a NOK if it doesn't receive first Snapshot from the host within this time.*

SCI FW will compute timer value for the next expected snapshot using the following formula:

- *Snapshot interval Timer in SCI = T_{MAX_ODCC} +* σ (T) – T_{ISI_MAX}, where is calculated as
- *σ (T) = T_{CYCLE}* – time difference between current and previous Snapshot received + σ (T-1).
- *For the first Snapshot, σ(0) is computed as:*
- *σ(0) = $T0_{IDEAL\_ODCC}$* - (Timer shapshot on receiving the first snapshot – Timer snapshot on receving Start Workload command)
- *Cross_Channel_Timer = CROSS_CHANNEL_DRIFT + σ (T)*

The deviation computed above could be negative if results arrived little later than the expected time (but before the maximum tolerance time).

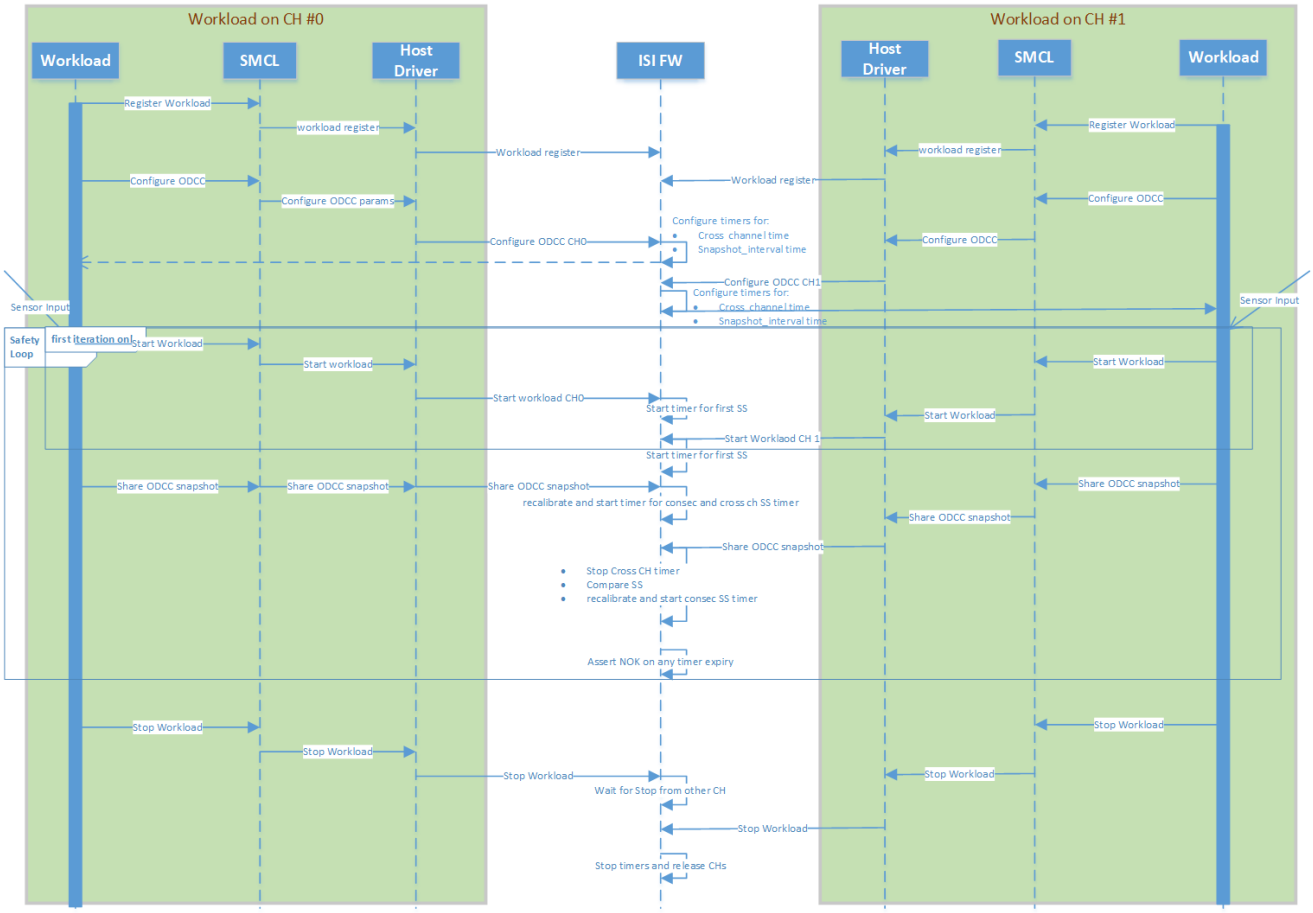
Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 972062	Accepted		SIL 3	SCI FST will ensure snapshots arrival time for a given safety loop iteration is between T_{MAX_ODCC} and T_{MIN_ODCC}. Otherwise, it will treat it as ODCC Failure.		N/A	Test	Intel Confidential
SSA_ 972023	Accepted		SIL 3	For Host application to start ODCC comparison across channels whether in 1oo1 or 1oo2D configuration, SCI provides following commands: i. Register workload – Register the safety application with the SCI FW. SCI FW will assign a unique Workload ID to the calling application. ii. Configure ODCC workload – This command will configure various timers for ODCC comparison that the ODCC FST will use with respect to this workload: 1. T_{CYCLE} - Safety Loop Time		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				2. T_{MAX_ODCC} - Maximum snapshot interval time 3. T_{MIN_ODCC} - Minimum snapshot interval time 4. CROSS_CHANNEL_DRIFT 5. $T0_{IDEAL\_ODCC}$ - Ideal time difference between Safety Loop invoking Start Workload and sending the first Snapshot 6. $T0_{MAX\_ODCC}$ - Maximum time difference between Safety Loop invoking Start Workload and sending the first snapshot iii. Start Workload – This command is an indication to SCI that the safety loop has started and SCI FW should expect the ODCC Snapshots (and host STL results if that is also included in the safety loop) from the workload from this point onwards. This Command is sent only during the first safety loop execution and includes a bitmask of what functions will run as part of safety loop. Currently only Host STL and ODCC are the only options supported. iv. ODCC Snapshot send – This command will allow Workload to send the snapshot to the SCI FW. If this is the first snapshot, SCI FW will stop the first snapshot timer and will compute the deviation and will calibrate the snapshot interval timer value for the current CH and will start the consec timer and cross channel timers using this. Else, if the snapshot from the other channel is available, it will perform the comparison and will recalibrate and start the snapshot interval timer. If any of these timer expires or the snapshot comparison fails, SCI FW will assert NOK. SCI FW will also check the snapshot ID for each packet to ensure that no snapshot from any channel is missed. Otherwise, SCI FW will assert NOK. v. Stop Workload - This message is				

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				sent by safety application when it wants to stop a periodic ISI function (Host STL or ODCC) execution. SCI will no longer expect the corresponding results from the Host. SCI will drop any snapshots that are received after it has received 'Stop Workload' command for a workload.				

4.3.5.1 1oo1 configuration's ODCC end-to-end flow

Figure 42 ODCC end to end flow in case of 1oo1 configurations



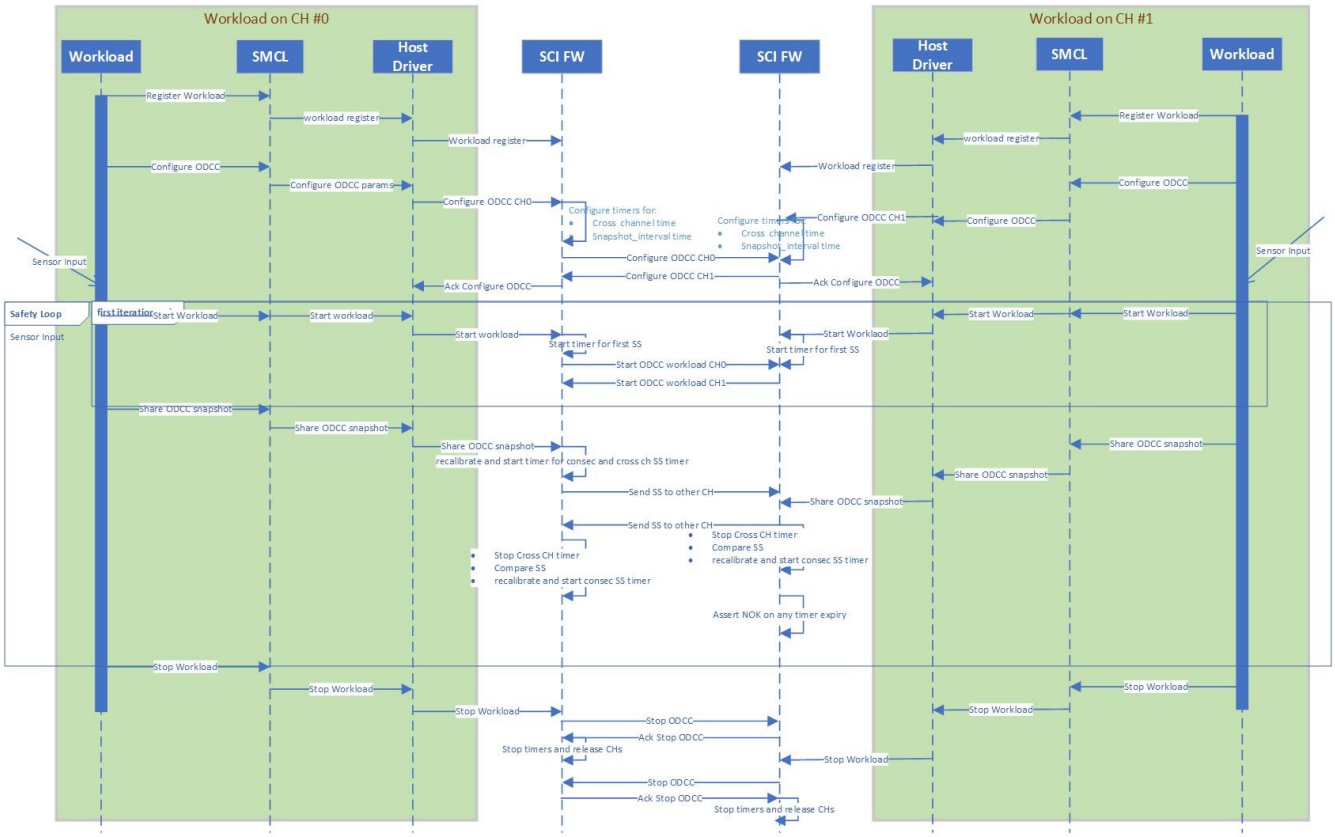
Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590519	Accepted		SIL 3	• As the ODCC functionality is done through workload code modifications, snapshot is computed at pre-determined		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				points in the workload and hence expected to generate the identical ODCC snapshot. Difference in snapshot will indicate a fault. • The SCI FW will process the messages from the SCI host driver for each of the ODCC pair. Compare the snapshots from two cores (see below for more details) and issue OK/NOK accordingly • If SCI FW does not receive any ODCC snapshots for any interval, as configured in the ODCC configure command, from any instance (workload had some issues and didn't get to send the snapshots to SCI) – then generate a NOK condition. • If the SCI FW detects a missing snapshot generated by a given workload (for example, SCI FW receives snapshot from a given workload with snapshot ID = 4 and the next snapshot has snapshot ID = 6. Snapshot ID of 5 is dropped), the SCI FW should generate NOK. • Due to drift associated with two instances of the WLs, snapshots will come in staggered fashion and SCI ODCC FST will need to account for that. • For every snapshot that is saved for further processing, ODCC FST will start a timer of value "cross_channel_drift" to ensure that the second channel sends the same snapshot before this timer expires. If this timer expires, SCI FW will assert NOK. • SCI also checks that 2 consecutive snapshots from a single channel are not more than the computed snapshot interval time as mentioned above. • If any of the above criteria is not met, SCI FW will assert NOK There could be multiple such workloads running in parallel on EHL within certain permissible drift, each using two cores. The SCI driver will be responsible for taking these snapshots from all the workloads and send them to SCI FW. Above diagram and description assumed single workload running on two cores. However one can simply replicate that setup for multiple workloads.				

4.3.5.2 1oo2D configuration's ODCC end-to-end flow

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590521	Accepted		SIL 3	The end-to-end flow diagram for 1oo2D cross monitoring configurations is given below. This is similar to 1oo1 ODCC setup above, but the safety workload is executed in two different EHL channels (instead of two cores of same EHL). Most of the content provided in previous section still applies for 1oo2D cross monitoring configuration and let's identify the differences from SW/FW flow point of view between these two configurations: 1. SCIs of two channels are cross connected to exchange messages. This cross connection is used to exchange the snapshots so both SCIs can do the comparison and find out faults based on cross comparisons. 2. For every EHL channel, the workload is executed only once and hence SCI gets snapshot from only the core executing the workload. No significance of core ID in this case. 3. It is SCI's responsibility to send the received snapshot to the pairing SCI so that each of the SCI instances get two copies of the snapshot. 4. SCI FW on both the channels verify the FW version running on other channel when the cross channel comparison is setup for the first time after boot and inform their respective hosts if there is a mismatch. 5. If SCI FW on one channel receives NOK and NOK reason is ODCC, then the working channel will also assert its NOK. 6. SCI to SCI communication is over SPI and hence it is even more importance to reduce the amount of data sent across to a minimum. The format for the SCI-SCI inter-communication packet is provided in figure below.		N/A	Test	Intel Confidential

Figure 43 ODCC end-to-end flow in case of 1oo2D cross monitoring



Here are some observations from the figure above:

- SCI will compare the snapshot only after it has both the snapshots available - local ones from the host and the remote ones from peer SCI.
- Both the SCI instances are comparing the snapshots from each other to ensure results are available with both channels.

4.3.5.2.1 1oo2D resumption of ODCC flows on a faulty channel replacement

In 1oo2D, a scenario may arise when one of the channels develops a non-ODCC fault. In that case, the SCI on the faulty channel will inform SCI of the other channel about the fault. SCI on working channel will send a NOK/Warning notification to its host, stop the odcc comparison mechanism and will flush out any pending snapshots for comparisons. It will ignore any shapshots that may come from host application.

It will also disable the CHX_RELAY_SWITCH so that it no longer monitors the OK/NOK pins of the faulty channel. From this point onwards, working channel works in single channel mode.

Customers may choose to restart the faulty channel or replace it with a new channel. Once they do that, they may want to restart the ODCC mechanism without stopping/restarting the working channel safety loop.

Hence, we define the following AoUs for the customers to take care of:

- **AoU6:** Once the faulty channel is replaced, customers will make sure that the request SCI to reconnect CHX_RELAY_SWITCH from the working channel SW application so that SCI can start monitoring OKNOK pins of the other channel
- **AoU7:** To resume the ODCC flow, customer should make sure that both the channel safety loops invoke START_WORKLOAD command to let SCI know about their starting the ODCC mechanism.
- **AoU8:** Customer SW will take care of synchronizing the safety loop start across the two channels.

4.3.6 ODCC Snapshot Payload

Each ODCC snapshot will include the following:

- Workload ID (8-bits) à this field needs to be unique per workload so that SCI can differentiate between workloads.
- Core ID/ChannelID (8-bits) à this field indicates the core on which the workload is running on. An option here is to map APIC ID of the core to a zero-based value. This is done to reduce the size of snapshot payload.
- Snapshot ID (8-bits) à the snapshot ID is a constantly incrementing number for a given workload. This is used to compare the signatures from the two duplicated workloads. The snapshot ID wraps to 0 on reaching 256.
- Timestamp (16-bits) à This field will be reserved and can be used for sending timestamp in future.
- Type (8-bits) -> this type field indicates the type of the current ODCC snapshot. In current version only one type is supported, so the value will be always set to 1.
- Signature of snapshot (32-bits) à the signature is the CRC32 value of the snapshot (for all variables that are expected to be identical across two instances of the workload). This field is applicable for Type 1 ODCC snapshot data only. It is possible for the customers to override the CRC-32 algorithm with an equivalent signature generation algorithm.

The Type 1 ODCC snapshot payload format is as provided in below.

Table 14 Format for ODCC snapshot payload

+0	+1	+2	+3
ChannelID	WorkloadID	SnapshotID	Reserved
Reserved			
Signature			

The SCI FW is NOT aware of exact significance/semantics of sent data and it is only responsible for comparing the data and performing fault indication.

4.3.7 Host Software Stack Details

The roles and responsibilities of the host software stack are provided in this section. Additional details on the stack organization and flows are discussed in detail in Section “Host side SW Architecture”.

The host software stack provides the following functionality w.r.t. ODCC feature.

- Provide a user space level ODCC library which provides the variable patrolling API for variables which require zero drift as well as for variables which require tolerance.

- Provide a mechanism to send ODCC Start and ODCC Stop messages to the SCI.
- Provide a mechanism to send the snapshots to the SCI.

4.3.8 SCI FW Flow Details

The ODCC FST which is a part of the SCI FW performs the following steps:

- Register the ODCC FST callbacks with FSTM for 'Configure workload', 'Start Workload, 'Stop Workload' and 'ODCC Share Snapshots' messages over PCIe MB in case of 1oo1 and 1oo2D configurations.
- Register the ODCC FST callbacks with FSTM for 'Configure workload', 'Start Workload, 'Stop Workload' and 'ODCC Share Snapshots' messages over SPI in case of 1oo2D configurations.
- In case any of the timer expires or if snapshot comparison fails, NOK will be asserted.
- On the receipt of 'Share Snapshots' message, perform the following:
    - Find the corresponding snapshot (based on Workload ID and snapshot ID) from the other channel. If not found, store this snapshot in local memory for further processing.
    - Compare signature
    - Make sure core/channel ID and workload id is as expected.
- Transfer the snapshot to the peer SCI in case of 1oo2D configurations.

4.3.8.1 Handling failure in a single channel of 1oo2D configurations

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 590547	Accepted		SIL 3	In case of 1oo2D cross monitoring, it is possible for one channel to fail due to thermal reasons, un-correctable memory error, fatal error, STL fault, etc. With ODCC working across the two channels, this would result in an ODCC failure in the peer channel as well. This leads to the SIL3 compliant system to fail completely which is undesirable. It is not possible to identify the failed channel in the case of ODCC failures. However, for other failures such as fatal errors, STL faults, etc. it is straightforward to identify the failed channel. Subsequently, the peer channel is notified on the non-ODCC failure which causes the ODCC comparison to be suspended immediately on the receipt of the NOK message. This provides a way to keep the non-failing channel functioning till a replacement of failed channel happens.		N/A	Test	Intel Confidential

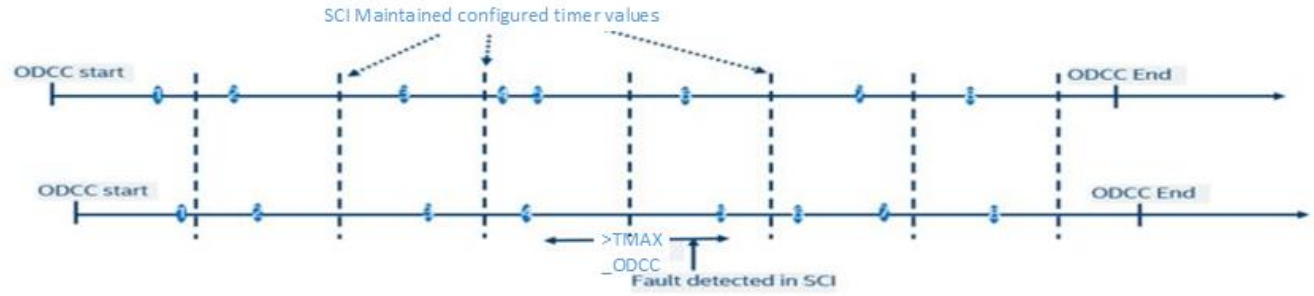
4.3.9 ODCC Failure Scenarios

There are multiple scenarios where SCI ODCC FST will return failure by generating a NOK. Here are the currently planned failure scenarios:

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 972064	Accepted		SIL 3	1. The drift between two channels cannot be more than cross_channel_drift. For every snapshot that is stored for further processing, we ensure the other channel sends a snapshot before this time. This is illustrated in the following figure:		N/A	Test	Intel Confidential

Figure 44 ODCC failure - high drift between snapshots



Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 849289	Accepted		SIL 3	In this case of snapshot #5 from second channel arrives later than cross_channel_drift time from reception of snapshot #5 from first channel, SCI detects this as a fault. 2. Apart from above, SCI also monitors that consecutive snapshots of a single channel are not apart by time specified by customer in start_workload API. In this case of snapshot #6 from channel1 arrives later than Maximum snapshot interval time (assuming that was specified in the configure_odcc API) from reception of snapshot #5, SCI detects this as a fault. 3. Regular fault scenario is shown in the following figure		N/A	Test	Intel Confidential

Figure 45 Regular fault scenario

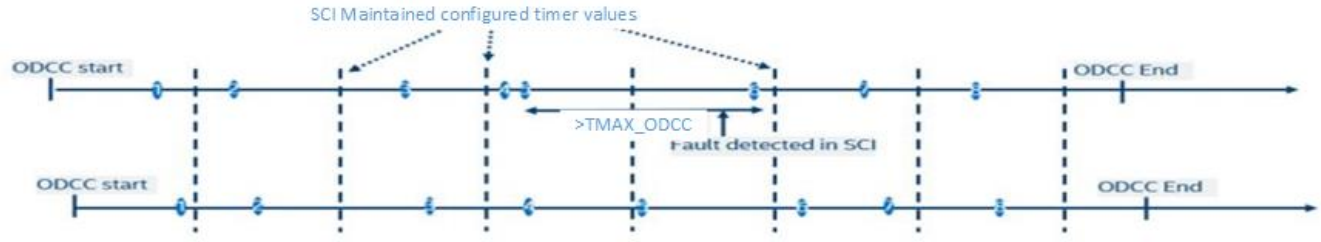
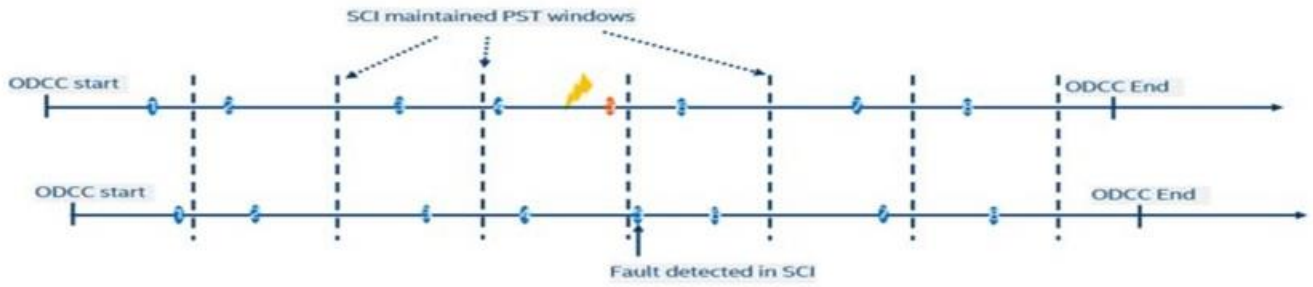


Figure 46 ODCC failure - snapshot comparison failure



Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 972063	Accepted		SIL 3	4. If the snapshots received from both the channels do not match.		N/A	Test	Intel Confidential

Non failure scenarios: It is ok to get snapshots from two channels that will span two PST windows as long as above conditions are met.

4.4 Proof tests flow for 1oo2D cross monitoring configuration

Proof test is necessary to detect dangerous hidden failures in a safety related system so that, if necessary, a repair can restore the system to 'as new' condition or as close as practical. Proof tests are primarily applicable only for 1oo2D configurations. The test is performed by CSE and SCI just performs the necessary orchestration. Hence there is no FST in SCI for this. FSTM does the job of the to and fro messages transfers from master and slave channels and required orchestration.

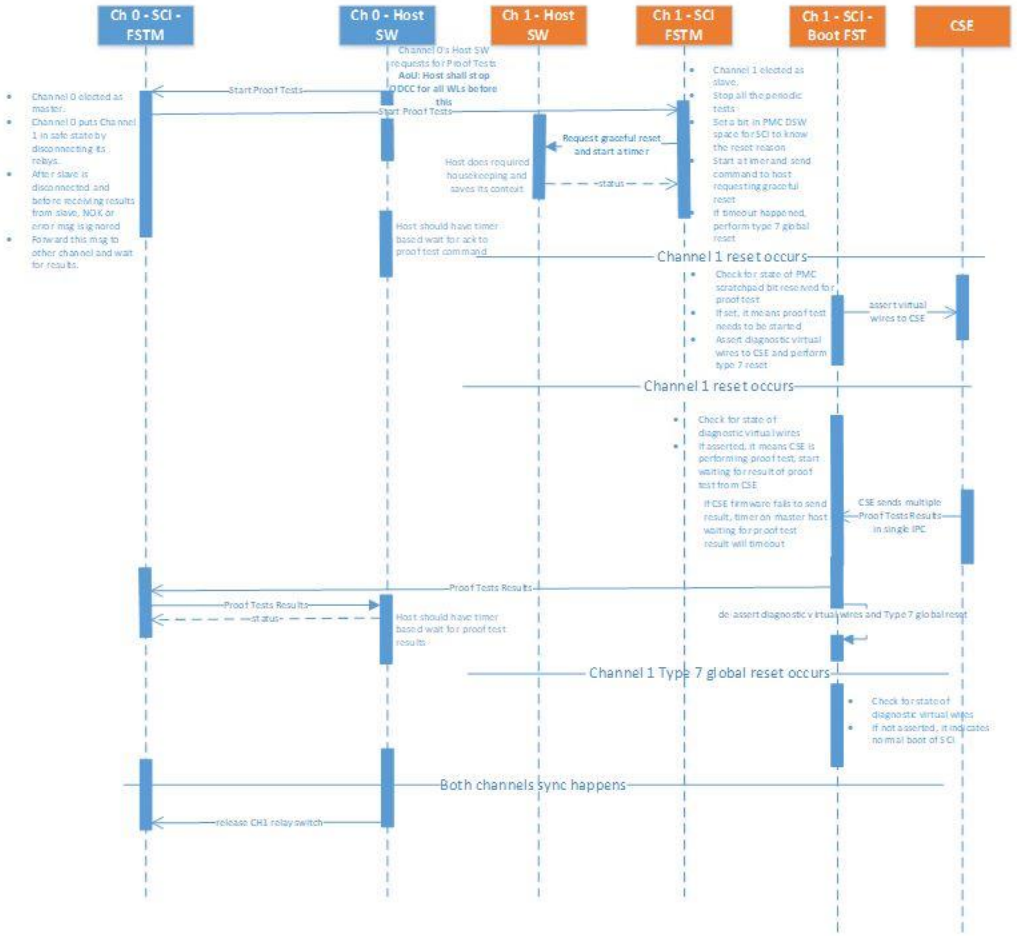
The channel on which proof test is executed is called as slave channel and the other channel is called as master channel. At any time, only one channel of the 1oo2D configuration will be under proof tests (slave channel). The system will be operating with only single channel (master channel only) during this proof testing time. This is to meet availability goal (zero downtime) of the system. System integrator has to make sure, the proof test is not triggered simultaneously on both channels simultaneously. If another proof test is triggered while proof test in progress, it would be rejected.

High level requirements for Proof tests

1. Only one channel of the 1oo2D configuration will be under proof tests at any time (slave channel). It is system administrator responsibility to ensure this criterion is met.
2. The master channel is responsible for putting the slave channel in a safe state before the execution of proof tests.
3. The results of proof tests are communicated to master channel after the execution of proof tests. In case of proof test failures, the slave channel will remain in safe state.
4. ODCC FST shall be stopped both on master and slave when proof test is started.

Proof Tests Flows

Figure 47 Proof Tests Flows



Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850134	Accepted		SIL 3	**Proof test flow 1002D: Master channel triggers the Proof test flow** 1. On receiving the 'START PROOF TEST' command from its local host software stack, SCI of that channel becomes 'master channel'. The other channel becomes the 'slave channel'. Note that the master channel flow can also be triggered by an MCU in 1oo1S system. 2. FSTM interface thread in master channel de-energies the relays of the slave channel by driving the CHX_RELAY_SWITCH signals. The slave channels is now in safe state. 3. FSTM interface thread in the master channel forwards the 'START PROOF TEST' command to slave channel. 4. After slave relays are disconnected and before receiving results for proof test, any message from slave shall be ignored. That means SPI interface thread waits indefinitely		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				for proof test results and ignores any other input. The related safety application will wait for a timeout to receive the proof test results. 5. After getting result from slave, forward it to host. **Slave channel proof test execution** 1. FSTM interface thread of the slave channel on receipt of 'START PROOF TEST' command, marks itself as 'slave channel'. 2. It stops all periodic FSTs, sets a bit in DSW scratchpad area and requests host for graceful shutdown. 3. On reboot, read the scratchpad bit that was set in last boot. If set, set virtual wire to CSE and perform global reset through PMC. 4. There are 2 events that happen on this reset, a. The CSE FW resets and runs the required proof tests. The proof test results are communicated to SCI using the CSE to SCI IPC mechanism. b. The SCI FW also resets and during its boot flow, the startup test on reading the state of diagnostic virtual wires, loops waiting for proof test results from CSE FW. 5. On receipt of the proof test results from CSE FW, the startup test of the slave channel SCI, asserts the ALRET pin and signals the master channel (or MCU in case of 1oo1S) of poof test result to be sent over SPI and forwards the same (by packing it in a 'PROOF TEST RESULTS' message) to master channel (or MCU in 1oo1S case). The master channel then informs the host software stack on the proof test results. **Slave channel reboots and ODCC restoration** 1. The startup test then de-asserts the virtual wires Type 7 global reset is then requested. 2. There are two events which happen on this reset, a. Host CPU boots to OS level and WL restoration need to take place so that ODCC can be restarted. As mentioned above, this is an AOU for EHL. b. The SCI FW also resets and the startup tests of slave channel on reading the state of diagnostic virtual wire finds that it is normal boot for SCI. 3. While in proof test mode, Slave channel won't accept any new START PROOF TEST request from MCU/Master channel				

AoU9: All WLs participating in ODCC shall send ODCC stop before SCI can get start proof test command

AoU10: Host performs sync between master and slave channels after the end of proof test

AoU11: After performing the sync between master and slave channels, host on master channel sends command to SCI to remove the signal that de-energized the relays on slave, thus bringing slave out from safe state.

4.5 SCI STL FST

This FST is responsible for periodically running STLs on SCI. Following tests are run as part of this STL:

4.5.1 frCPU Specific STL

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590563	Accepted		SIL 3	As a part of periodic fRCPU test, the SCI STL FST shall perform the following operations: - Periodic SCI STL FST:- Multiplication and Division Test		N/A	Test	Intel Confidential
SSA_850168	Accepted		SIL 3	- Periodic SCI STL FST:- Bus Matrix Test. This test is run both for single port and double port memory to ensure that arbiter also gets tested. If the test fails, the FST will assert the NOK		N/A	Test	Intel Confidential
SSA_850169	Accepted		SIL 3	- Periodic SCI STL FST:- NVIC Interrupt Priority register test frCPU user guide/safety manual shall be referred for the implementation of frCPU specific STLs.		N/A	Test	Intel Confidential

4.5.2 FMM scratchpad register Test

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_850139	Accepted		SIL 3	Following steps are performed each time SCI internal FST is invoked: 1. Write 0xA5A5A5A5 into to FMM scratchpad register. 2. Read back scratchpad register. 3. Write 0x5A5A5A5A into to FMM scratchpad register. 4. Read back scratchpad register. 5. If value read after step 2 and step 4 is other than written value, the test reports failure.		N/A	Test	Intel Confidential

4.6 Platform Boot FST

Platform Boot FST is run at SCI boot. It is responsible for carrying out the following, and in case of any error, it invokes the required error handling

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_850142	Accepted		SIL 3	1. Running various SCI Boot Safety Tests. These include:		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				• Timer functional test(GP timer, periodic timer) • WDT functional Test • Timer fault injection test • Clock monitor fault injection test • Root parity fault injection test • FMM Scratch pad test • fRCPU busmatrix /SRAM test • frCPU MultDiv, NVIC interrupt priority parity, MPU and NVIC register read				
SSA_ 850143	Accepted		SIL 3	2. Host Boot monitoring - Boot Complete from BIOS - POSC test results(This also informs SCI about the number of physical cores on the platform		N/A	Test	Intel Confidential
SSA_ 850155	Accepted		SIL 3	3. Ping with Pcode over RAVDM and get ITD(inverse temperature dependency) parameters		N/A	Test	Intel Confidential

4. Ping with PMC using get platform transition information query command.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850157	Accepted		SIL 3	5. Startup STL results from Safety cores SCI sends PST and DTI values to host safety application running startup STL in response to STL results. If the test results indicate failure, SCI will assert NOK.		N/A	Test	Intel Confidential

6. All above commands and their response are blocking SCI and host to proceed with their boot flow. So proper command-response is mandatory for these commands. If host does not get ack from SCI for any command, it may resend the command even if SCI received the command and is expecting next command. In that case, if previous command is received again while next command is awaited, ack for that shall be sent to host.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 972065	Accepted		SIL 3	7. Compare ODCC snapshots coming from host test application if it comes while startup tests are executing. Host can use this to test ODCC software safety mechanism. Host SW may send ODCC snapshots with injected errors and Boot FST will invoke ODCC Snapshot comparison safety mechanism which should detect these injected errors and return appropriate status code to Boot FST, which in-turn, reports these statuses back to Host SW stack.		N/A	Test	Intel Confidential
SSA_ 850153	Accepted		SIL 3	Startup tests invoke error handling flow and will assert NOK based on any error encountered in Host Boot Tests and SCI Boot STL through return values of APIs. In case of success, control is given back to the FSTM to exit safe state During startup, SCI will receive the system time in UTC format from the host as part of host boot complete message. This system		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				time is used by diagnostic data service to timestamp the diagnostic data. SCI will store the RTOS timer and system time and provide them to diagnostic data service. Whenever an event occurs the time stamp for the event is calculated as, HostTimeStamp (UTC format) + UTC conversion of (SCI Current RTOS time – SCI HostBootComplete Message receive RTOS time) The following flow diagram details these steps:				

Figure 48 Platform Startup tests – SCI Startup tests

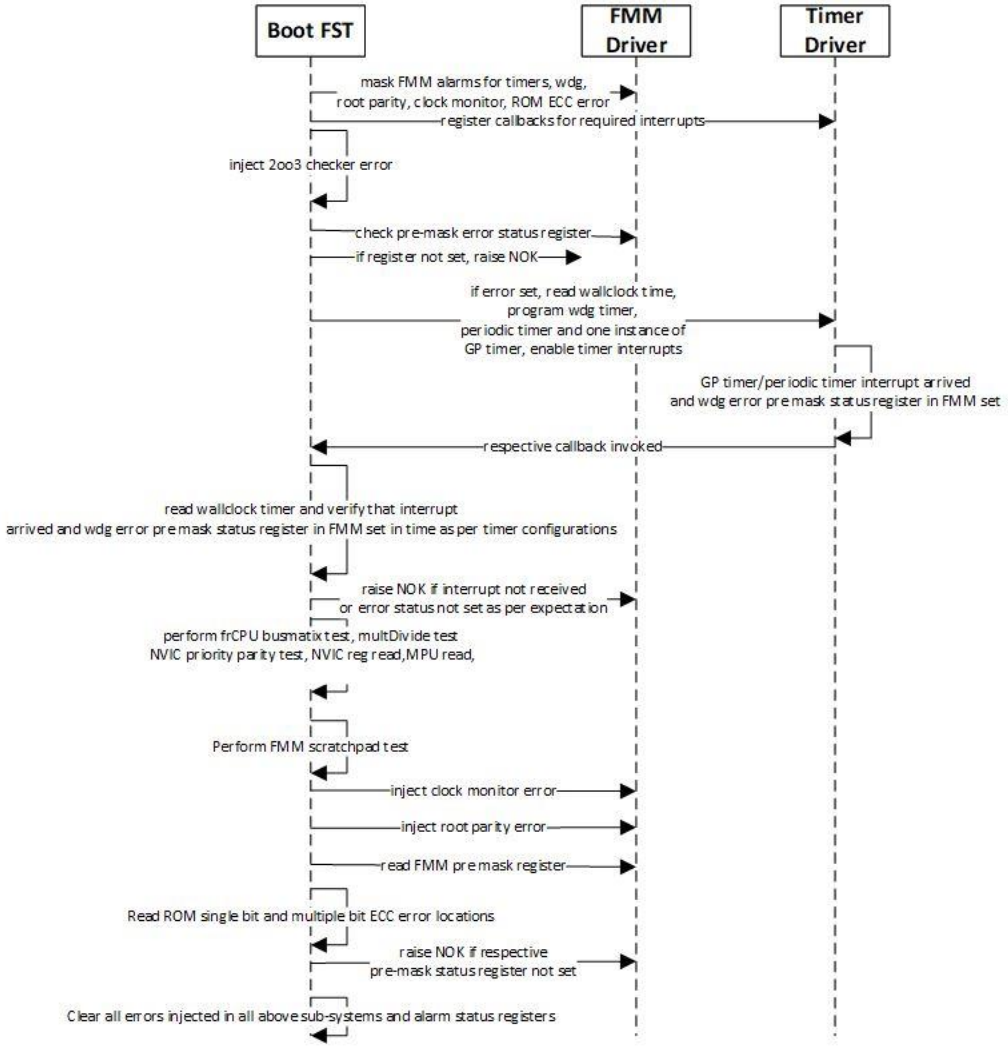
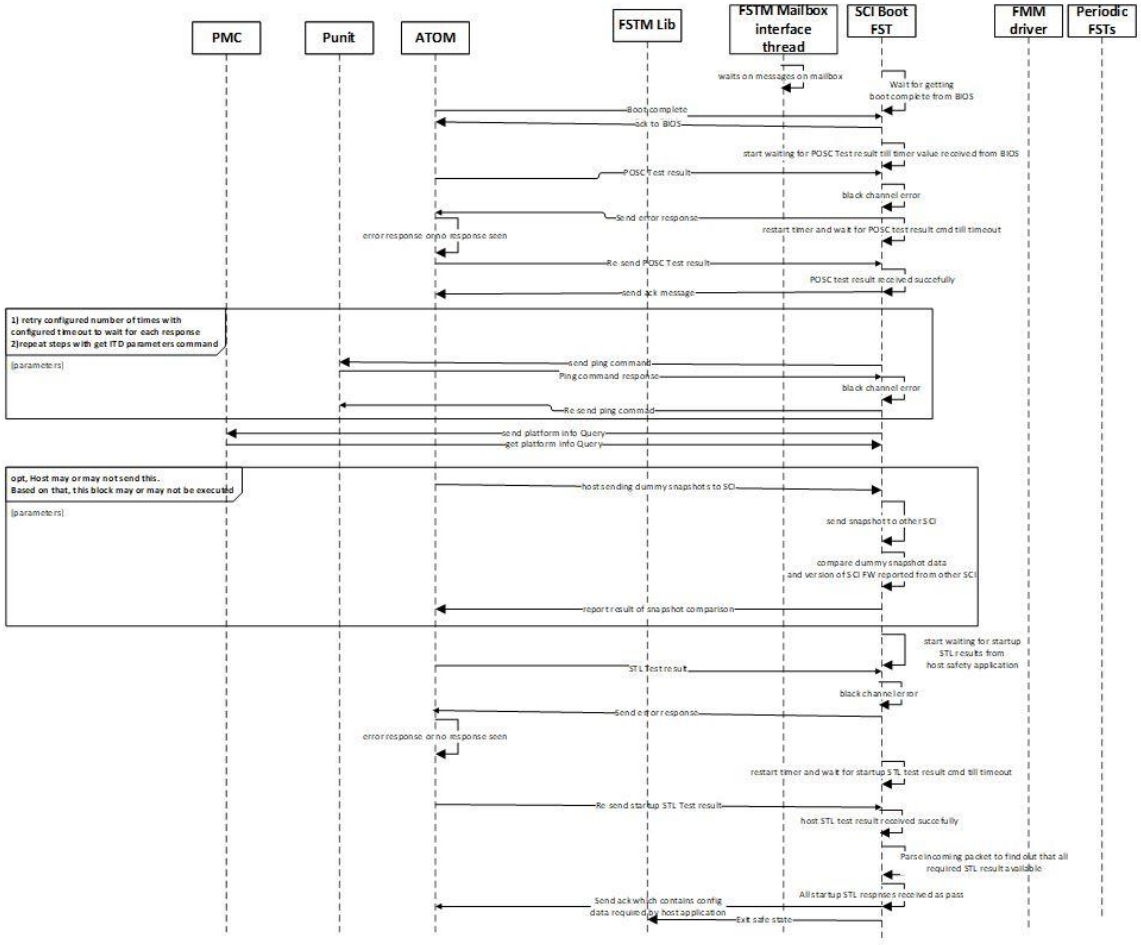


Figure 49 Platform Startup tests – Host Startup tests



4.7 Various Error Handling FSTs

1. Corrected Error Handler FST
2. Non-Fatal Error Handler
3. FATAL Error Handler FST

4.7.1 Platform Error Flows

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850172	Accepted		SIL 3	All the errors from platform components are routed to global IEH and SCI will monitor the errors from Atom and IEH. ERR_1 can be software recovered. If there ERR_1 is successfully recovered then SCI will not issue NOK, else ERR_1 is treated as error and SCI will issue NOK. The following table list the errors monitored by SCI.		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850173	Accepted		SIL 3	During boot phase, these errors will be unmasked by startup tests before switching to OK state.		N/A	Test	Intel Confidential

SCI will mask PROCHOT alarm before switching to OK state.

Table 15 Platform Error priority

FATAL Error with Error Diagnostic information (ERROR)	FATAL Error without Error Diagnostic information (ERROR)	Correctable Error (WARNING)	INFO
ERROR_1 (recovery failure) ERROR_2 CATERR_N fRCPU_ERR (all errors)	Parity Errors (all parity errors) DMA_LS_Alarm Fuse_CRC ROM_ECC_2Bit MBist_Alarm WDTimer_Alarm RTOS_Alarm RF_uerr RTOS_alarm RF_MBIST RAM_MBIST ROM_MBIST SRAM_ECC_2bit NOC_Fabric_FATAL IOSF_Bridge_unsupported	ERROR_1 (recovery success) ROM_ECC_1bit RAM_ECC_1bit RF_cerr NOC_Fabric_NonFatal	ERROR_0

4.7.1.1 Platform Error Priority

Table 16 Platform Error priority

Error Signal	Error Type	Priority
ERROR_1 (recovery failure) ERROR_2 CATERR_N fRCPU_ERR (all errors) Parity Errors (all parity errors)	FATAL	P0

DMA_LS_Alarm Fuse_CRC ROM_ECC_2Bit MBist_Alarm WDTimer_Alarm RTOS_Alarm RF_uerr RTOS_alarm RF_MBIST RAM_MBIST ROM_MBIST SRAM_ECC_2bit NOC_Fabric_FATAL IOSF_Bridge_unsupported		
ERROR_1 (recovery success) ROM_ECC_1bit RAM_ECC_1bit RF_cerr NOC_Fabric_NonFatal	WARNING	P1
ERROR_0	INFO	P2

4.7.1.2 Corrected Error Flow

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590570	Accepted		SIL 3	The corrected errors are signaled from multiple sources. Viz. 1. Corrected PCIe errors from PCH 2. Corrected errors from MCBanks 3. IO Fabric ECC_1bit error 4. RAM_ECC_1bit 5. ROM_ECC_1bit The firmware flow will be different for different configurations. Corrected errors are corrected by hardware and safety is not compromised due to corrected errors. The Predictable Failure Analysis (PFA) says, if there are too many corrected errors observed within short duration of time then the probability of hitting an uncorrected error is very high. Whenever there is an uncorrected error,		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				safety is compromised. The corrected error sources ROM_ECC_1bit, RAM_ECC_1bit, IO Fabric ECC_1bit along with other SCI warnings are grouped together and are counted. The total count is checked against threshold value. All the corrected errors reported through MCBank (signaled through CMCI) are grouped together and are counted. The total count is checked against threshold value. The error_0 and error_1 are two separate groups and the error_0 and error_1 reported by IEH safety devices are counted. The total count is checked against threshold value. All IBECC (Inband ECC) reported from host are grouped together and are counted. The total count is checked against threshold value.				
SSA_ 850179	Accepted		SIL 3	To address the issue hitting an uncorrected error due to frequent corrected errors, the corrected errors that occur within certain duration are counted. Whenever the count exceeds user configured threshold, the corrected error is upgrade to Warning Condition. The corrected error threshold is a configurable parameter for every corrected error. The value of threshold will decide SCI response to that corrected error. The correctable error threshold will be a configurable parameter.		N/A	Test	Intel Confidential
SSA_ 850180	Accepted		SIL 3	• Threshold = 0 è The error is never escalated to WARNING • Threshold = 1 è The error is escalated to WARNING every time • Threshold > 1 è The error is escalated to WARNING after error count is greater than or equal to configured threshold.		N/A	Test	Intel Confidential

The FMM can also be configured to escalate all corrected errors as FATAL errors, however at this time this feature is not supported on SCI.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850182	Accepted		SIL 3	The Predictable Failure Analysis says, the safety shall be compromised if there is a burst of correctable errors within a short duration. If the corrected errors are occurring over the period of time, then there is no threat of compromising safety. To prevent the corrected error from accumulating over period time and triggering a false alarm, corrected error count is decremented by one after certain duration until the error count reach zero. This mechanism is called as “Leaky Bucket”, where the errors are		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				collected in a bucket and are leaked based upon time. More information on Leaky Bucket Algorithm can be found at https://en.wikipedia.org/wiki/Leaky_bucket Refer FuSa User Guide for configuration parameters details.				
SSA_ 850183	Accepted		SIL 3	The duration at which SCI will decrement the accumulated corrected error count is a configurable parameter.		N/A	Test	Intel Confidential

4.7.1.3 1oo1 Basic Configuration Corrected Error Flows

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 590572	Accepted		SIL 3	1oo1 Basic Configuration don't have external Safety MCU. Diagnostic Data logs will be forwarded to host over mailbox. The below flow is when threshold is configured as 0 or 1 or n.		N/A	Test	Intel Confidential

Figure 50 1oo1 Basic High Level Corrected Error Flow

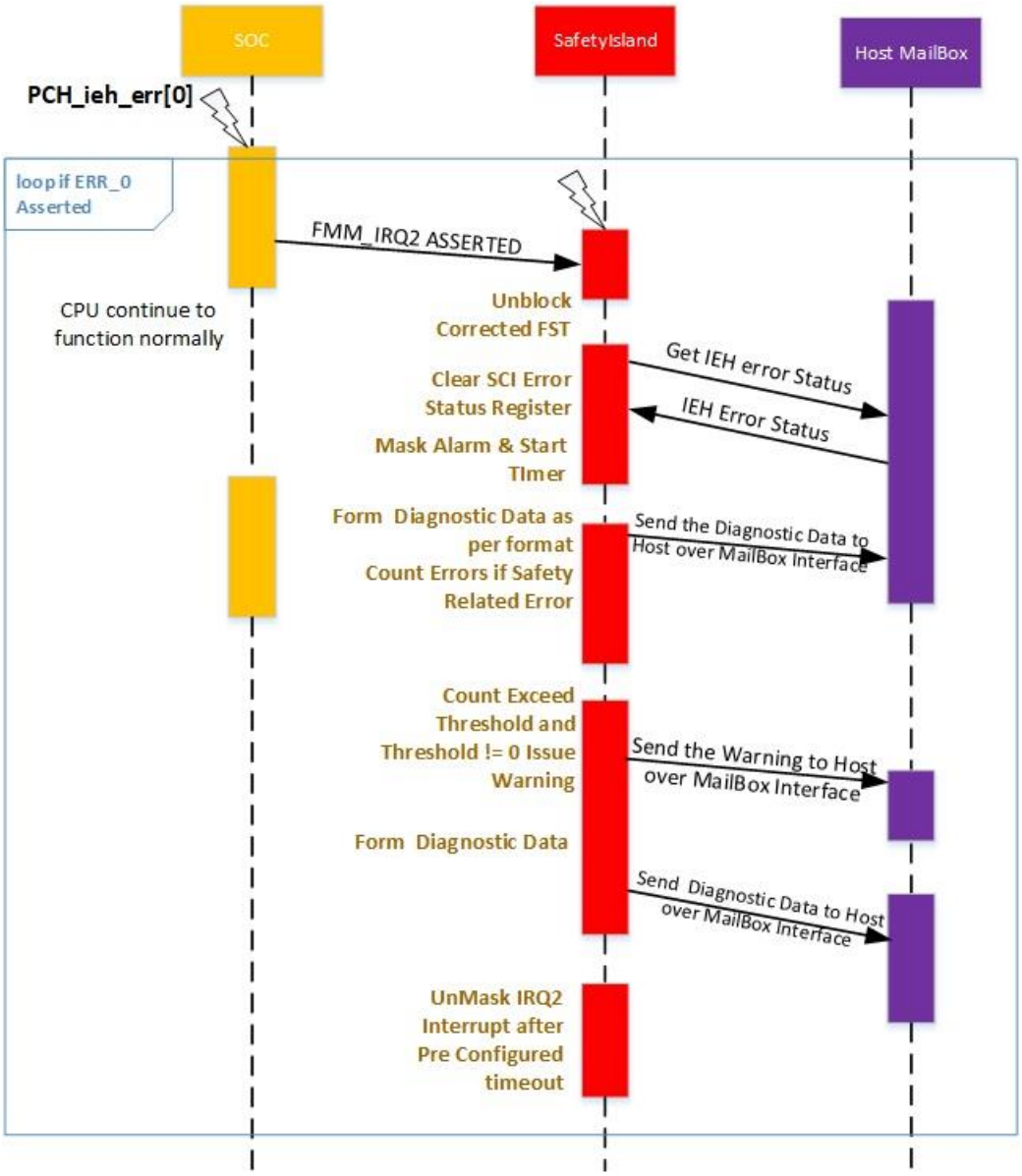
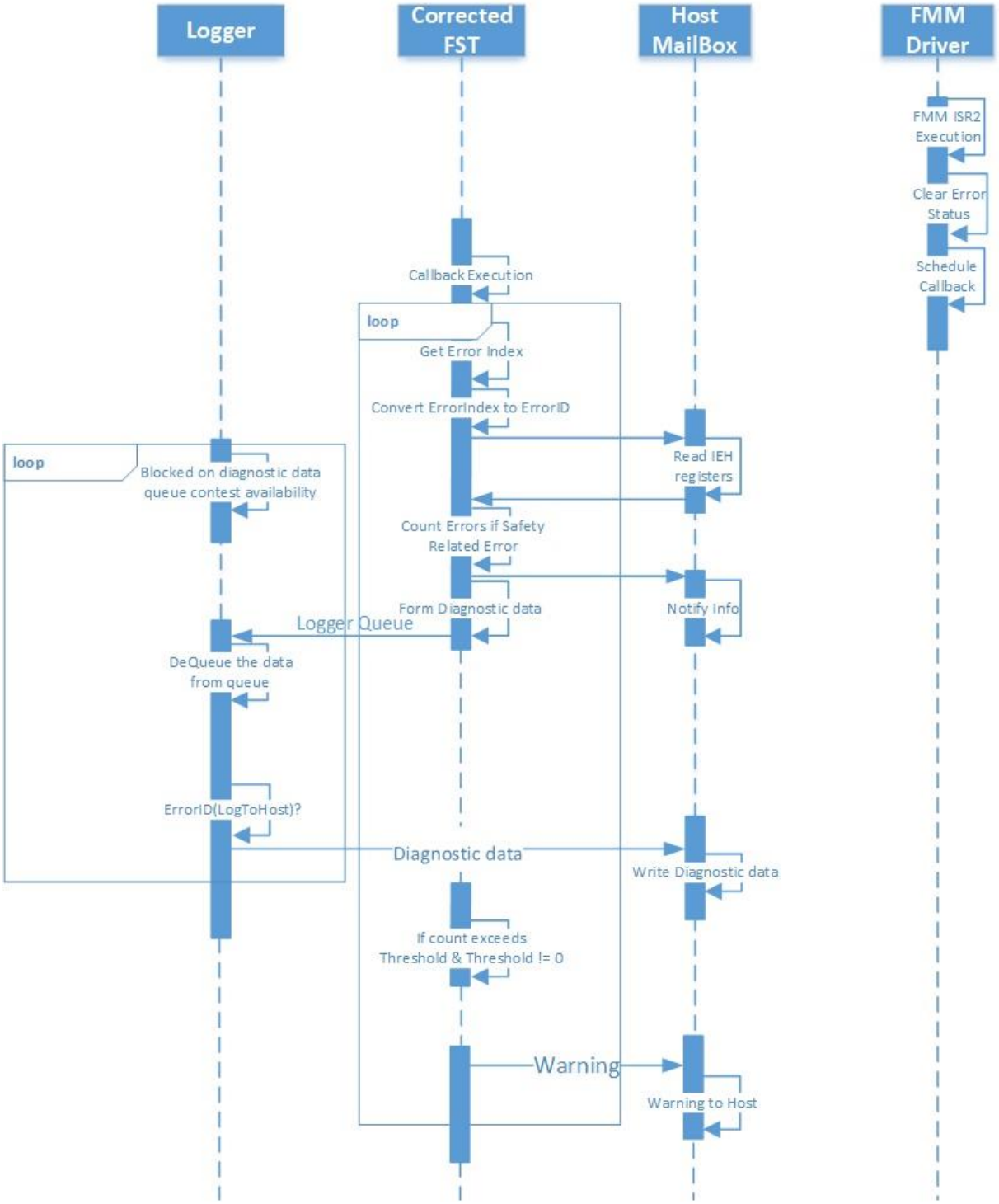


Figure 51 1oo1 Basic low level Corrected Error Flow



Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 590575	Accepted		SIL 3	1. 1oo1 Basic configuration don't have Safety MCU. So Error logs will be provided only to the host 2. In response to ERROR_0 ASSERTION, SCI hardware will update the error status register and trigger ISR2 3. The corresponding "error interrupt handler" will clear the status register and schedule a callback. The "interrupt handler" will also pass error index to the callback function. Callback function will unblock the corrected FST If number of corrected errors < Corrected_Error_Threshold, then return, create a log and send it to host 4. If number of corrected errors > Corrected Error Threshold, then notify NOK to host and also send the log to host 5. ERROR_0 can remain asserted for longer duration. To prevent same error_0 signaling SCI again, ERROR_0 will be masked for certain duration. 6. If corrected_error_threshold = 1, then send the log to host every time 7. If corrected_error_threshold = 0, then corrected error is never escalated to warning 8. Host processor will continue to run normally		N/A	Test	Intel Confidential

4.7.1.4 1oo1 Smart Configuration Corrected Error Flows

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 591336	Accepted		SIL 3	1oo1 Smart Configuration has external Safety MCU. Error logs will be forwarded to host over mailbox and also to the external safety MCU by asserting the ALERT Pin		N/A	Test	Intel Confidential

Figure 52 1oo1 Smart High Level Corrected Error Flow

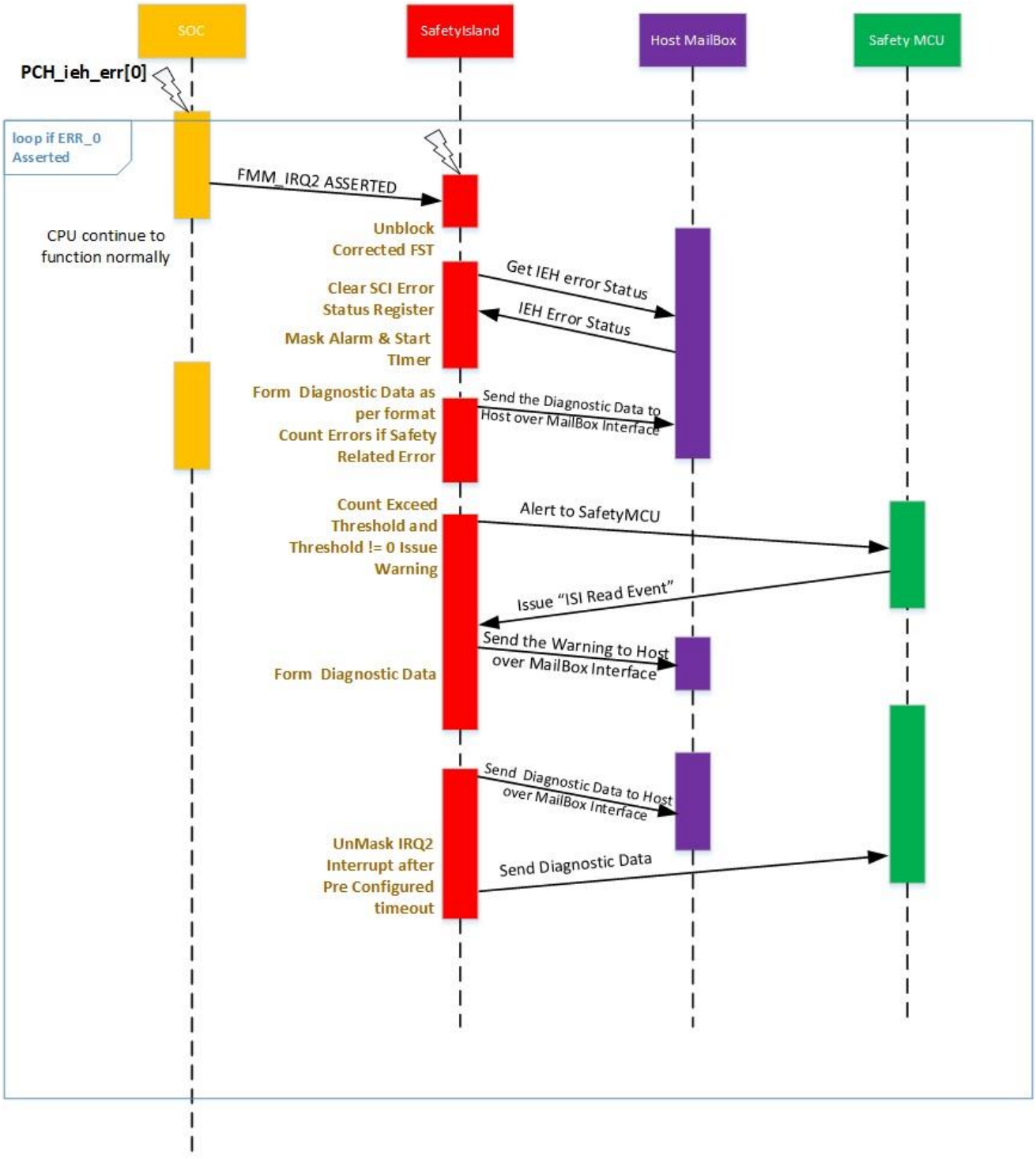
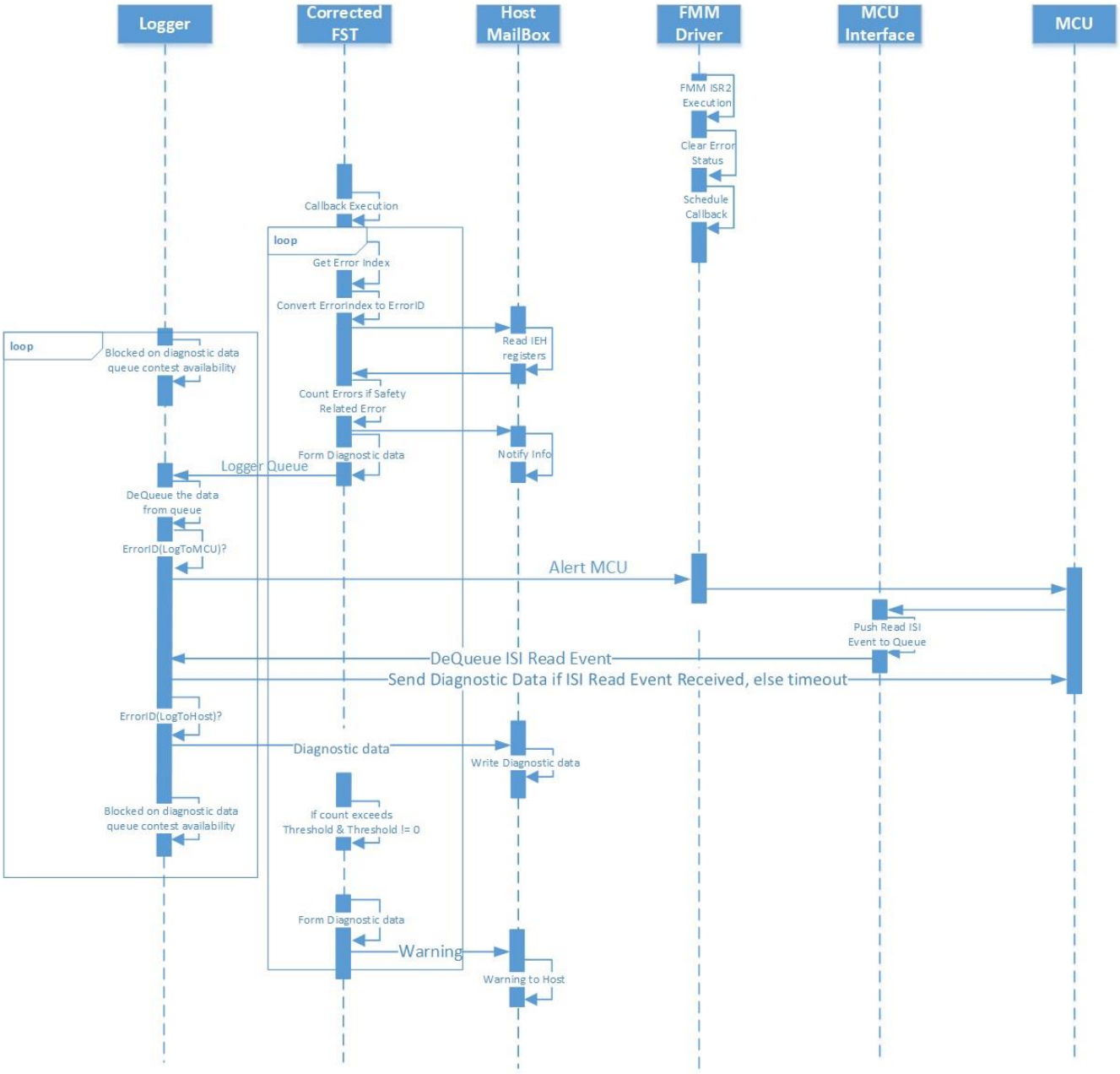


Figure 53 1oo1 Smart low level Corrected Error Flow



Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850192	Accepted		SIL 3	The SCI 1oo1 Smart flow for ERROR_0 is same as 1oo1 Basic flow, but the log will be forwarded to safety MCU until threshold. Once corrected error count exceeds, SCI will issue NOK to safety MCU and also forward		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				the error log. SCI will de-assert the alert, once MCU queries the error log.				

4.7.1.5 1oo2D Configuration Corrected Error Flows

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590584	Accepted		SIL 3	1oo2D Configuration has two channels. The first channel is called as monitoring and the second channel is called as monitored channel. Both channels act as monitoring and monitored channels for the other channel. For Channel X, Channel Y will act as MCU and vice versa. When there is an error in one of the channels, ISI on failing channel informs other channel about the error, ISI on the other channel queries the failing channel for diagnostic information and then sends that diagnostic data to its own host.		N/A	Test	Intel Confidential

Figure 54 1oo2D Corrected Error High Level Flow

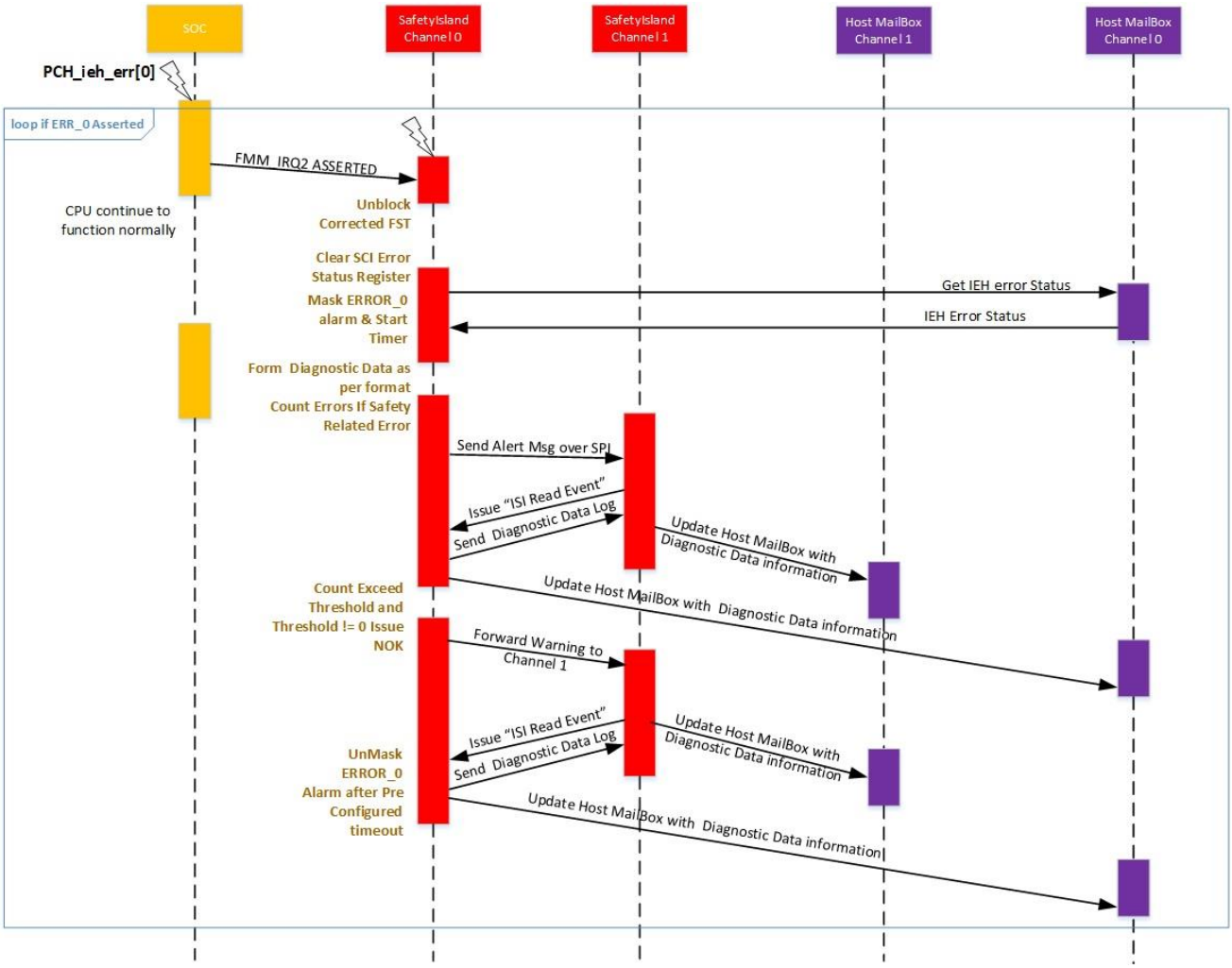
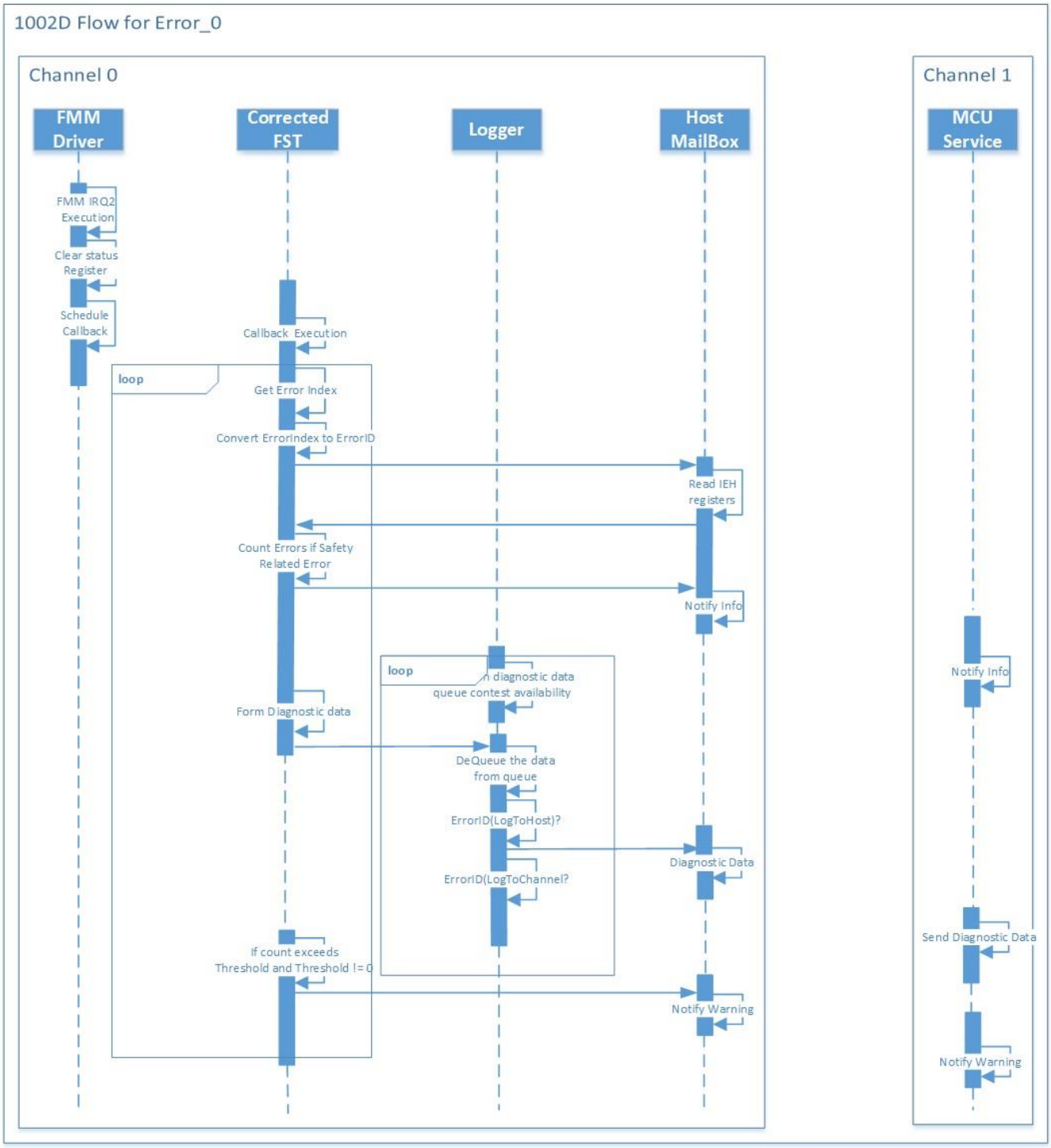


Figure 55 1oo2D Corrected Error Low Level Flow

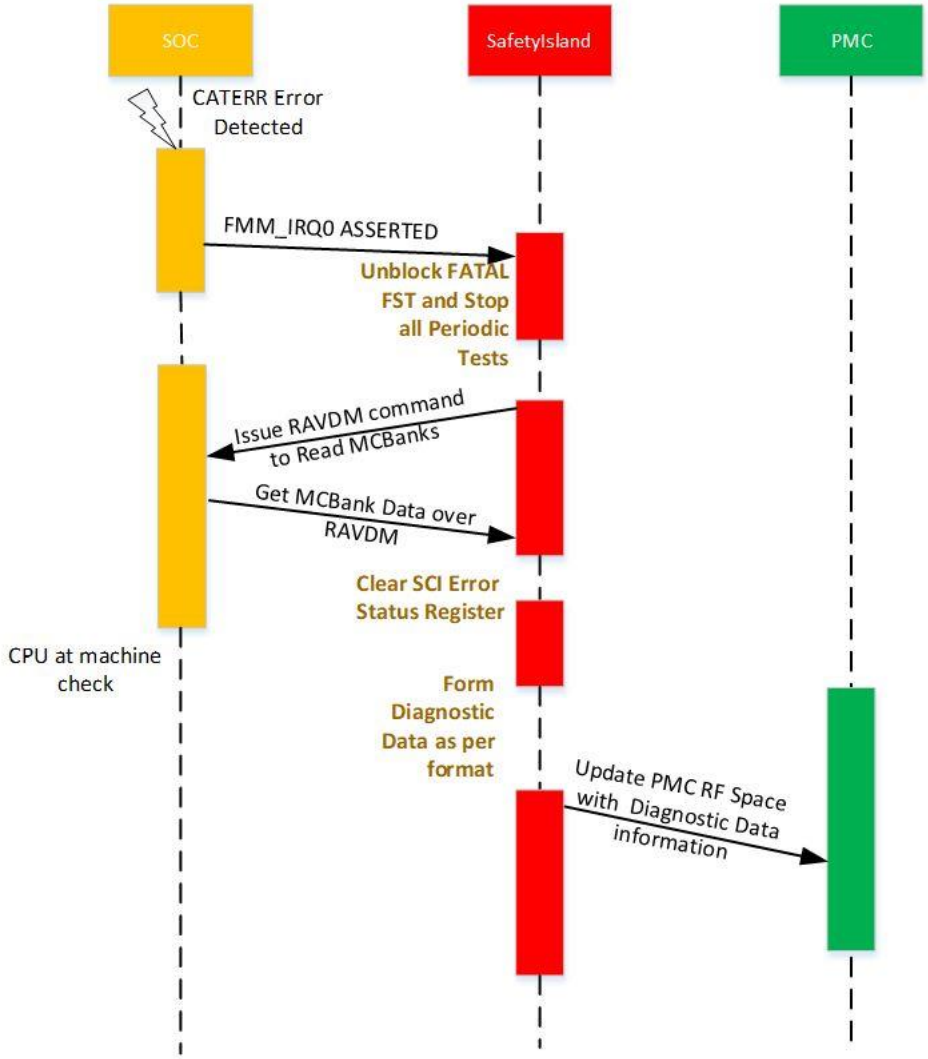


Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_	Accepted		SIL 3	**1002D Corrected Error Flow steps**		N/A	Test	Intel

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
590587				The 1oo2D corrected error flows are same as 1oo1 Basic flow, with following differences: 1. Whenever NOK is asserted SCI will notify the other channel the reason for NOK 2. Error log will be forwarded to other channel 3. Alert will be sent over SPI interface as a command to keep the error handling flows same between 1oo1 Smart and 1oo2D configurations.				Confidential

4.7.1.6 1oo1 Basic Configuration CATERR flows

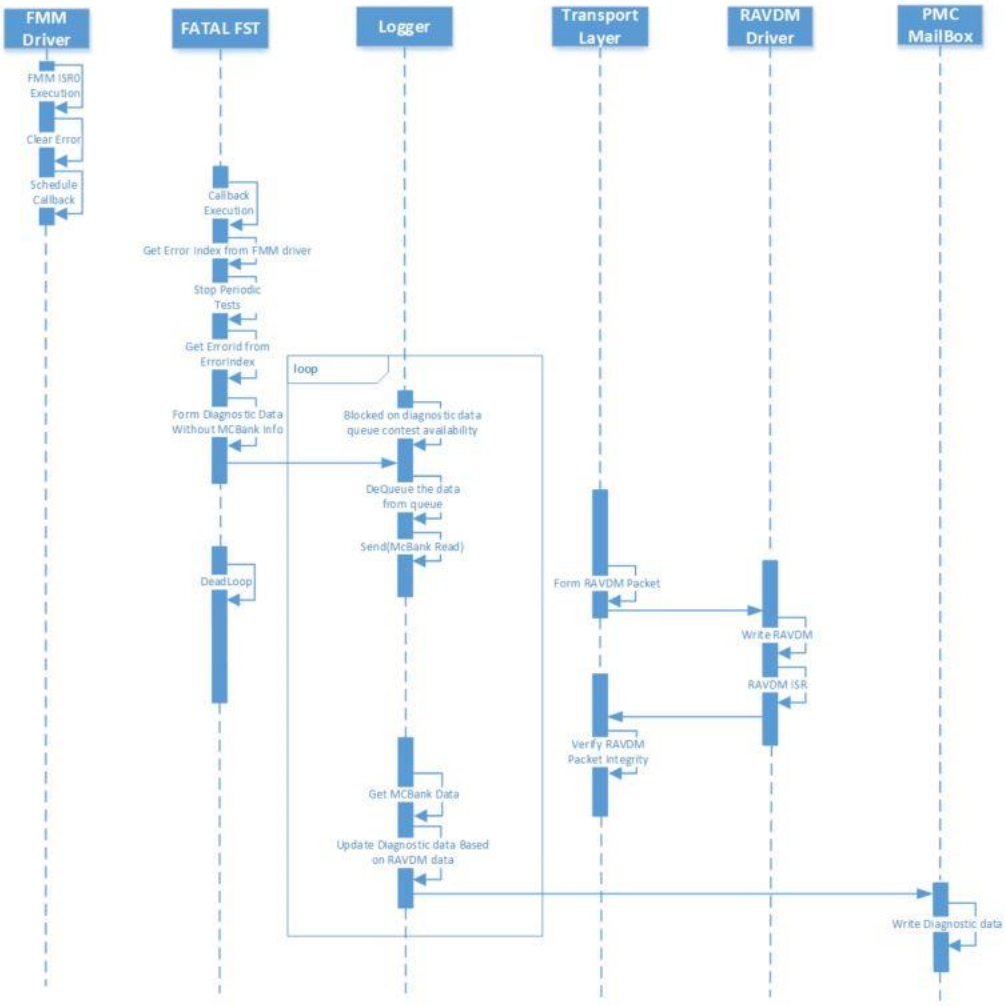
Figure 56 CATERR high level flow in 1oo1 Basic Configuration



CATERR is a FATAL error within processor CORE subsystem. When CATERR occurs.

ISI will query the MCBanks to find out the source of error. Due to limitation in RAVDM protocol, ISI cannot read the core MCBanks. The error log will have error source information for all SOC subsystem except core errors.

Figure 57 CATERR low level Error flow in 1001 Basic Configuration



1. Whenever there is a CATERR, FMM will assert IRQ0 and update the status register with the source of the error
2. FMM IRQ0 interrupt handler will clear the error schedule a callback and exit after passing the error id to the callback
3. 1oo1 Basic configuration don't have Safety MCU. So Error logs will be provided only to the host on next boot
4. Callback function will unblock the FATAL FST
5. FATAL FST will stop all periodic tests
6. FATAL FST will start a timer and issue a RAVDM MCbank read message
7. If the timer expires then the RAVDM read request will be cancelled and a generic error log without diagnostic information will be created.

8. If there RAVDM MCBank read is successful, then ISI firmware will stop the timer and create an error log with error diagnostic information.

9. FATAL FST will write the error log to PMC scratchpad area. On next reboot, Host software will issue a command to ISI firmware to provide the error log

10. Host software will write the error log information to host nonvolatile area

Note: ISI will not initiate a reboot upon CATERR. It is up to administrator on when to reboot the host.

4.7.1.7 1oo1 Smart Configuration CATERR flows

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590597	Accepted		SIL 3	The flow for 1oo1 Smart for handling the CATERR is exactly same as that of 1oo1 Basic, except the error log will be forwarded to Safety MCU.		N/A	Test	Intel Confidential

4.7.1.8 1oo2D Configuration CATERR flows

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590599	Accepted		SIL 3	The flow for 1oo2D for handling the CATERR is exactly same as that of 1oo1 Basic, except that on detecting the NOK, error log is forwarded to the other channel. The other channel will then query the failing channel for the Diagnostic information. Once received, it will send the diagnostic data to its own host.		N/A	Test	Intel Confidential

4.7.1.9 1oo1 Basic Configuration PROCHOT flows

SCI will mask PROCHOT in FMM and SCI won't take any action for PROCHOT signaling.

4.7.1.10 1oo1 Smart Configuration PROCHOT flows

SCI will mask PROCHOT in FMM and SCI won't take any action for PROCHOT signaling.

4.7.1.11 1oo2D Configuration PROCHOT flows

SCI will mask PROCHOT in FMM and SCI won't take any action for PROCHOT signaling.

4.7.1.12 1oo1 Basic Configuration IEH ERR2 flows

The error handling of IEH ERR2 for 1oo1 Basic is same as that of 1oo1 Basic CATERR handling, except IOSF SB interface will be used to perform error isolation.

4.7.1.13 1oo1 Smart Configuration IEH ERR2 flows

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590609	Accepted		SIL 3	The error handling of IEH ERR2 for 1oo1 Smart is same as that of 1oo1 Smart IEH ERR2 handling, except IOSF SB interface will be used to perform error isolation.		N/A	Test	Intel Confidential

4.7.1.14 1oo2D Configuration IEH ERR2 flows

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590611	Accepted		SIL 3	The error handling of IEH ERR2 for 1oo2D is same as that of 1oo2D IEH ERR2 handling, except IOSF SB interface will be used to perform error isolation. In addition, ISI will query the faulty channel ISI for Diagnostic data and will then send that diagnostic data to its own host		N/A	Test	Intel Confidential

4.7.1.15 1oo1 Basic Configuration IEH ERR1 flow

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590613	Accepted		SIL 3	ERR1 is an uncorrected Non-FATAL error where software can recover the system sometimes. The flows will differ based upon whether recovery is successful or failure. 1. 1oo1 Basic configuration don't have Safety MCU. So Error logs will be provided only to the host 2. In response to ERROR_1 ASSERTION, SCI hardware will update the error status register and trigger ISR1 3. The corresponding "error interrupt handler" will clear the status register and schedule a callback. The "interrupt handler" will also pass error number to the callback function. 4. Callback function will unblock the NON FATAL FST, and the FST will query the STL and check if recovery is successful. 5. If ERR_1 recovery is successful, then NON FATAL FST will query the IEH status registers over IOSF and form the log 6. The error log will be transferred to Host over mailbox 7. Upon exceeding the threshold, ERROR_1 is escalated to warning and the warning condition is notified to host 8. Host will continue to function normally.		N/A	Test	Intel Confidential

Figure 58 High Level ERR_1 Flow Diagram for Recovery Success

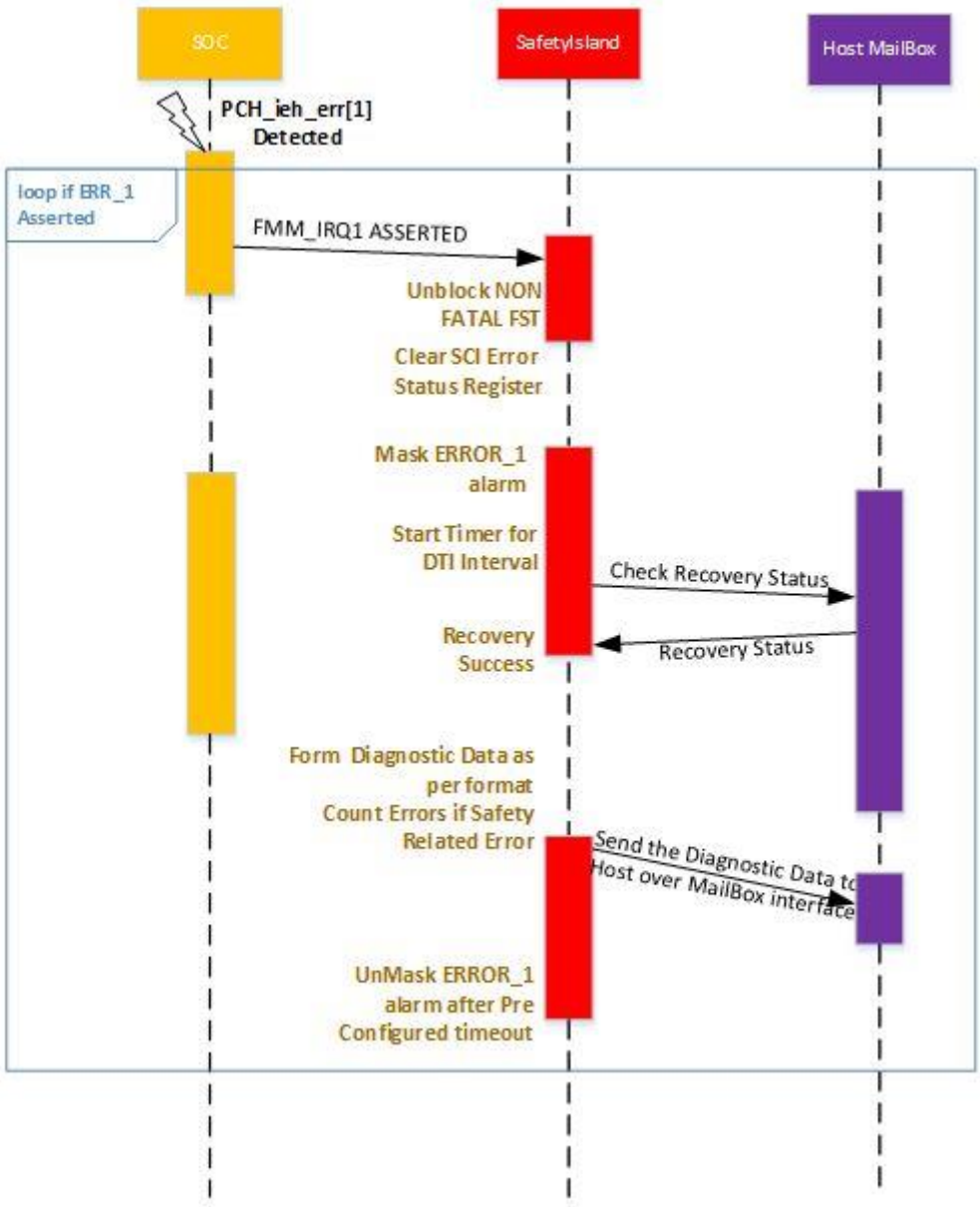


Figure 59 Low Level Flow Diagram for ERR_1 recovery success

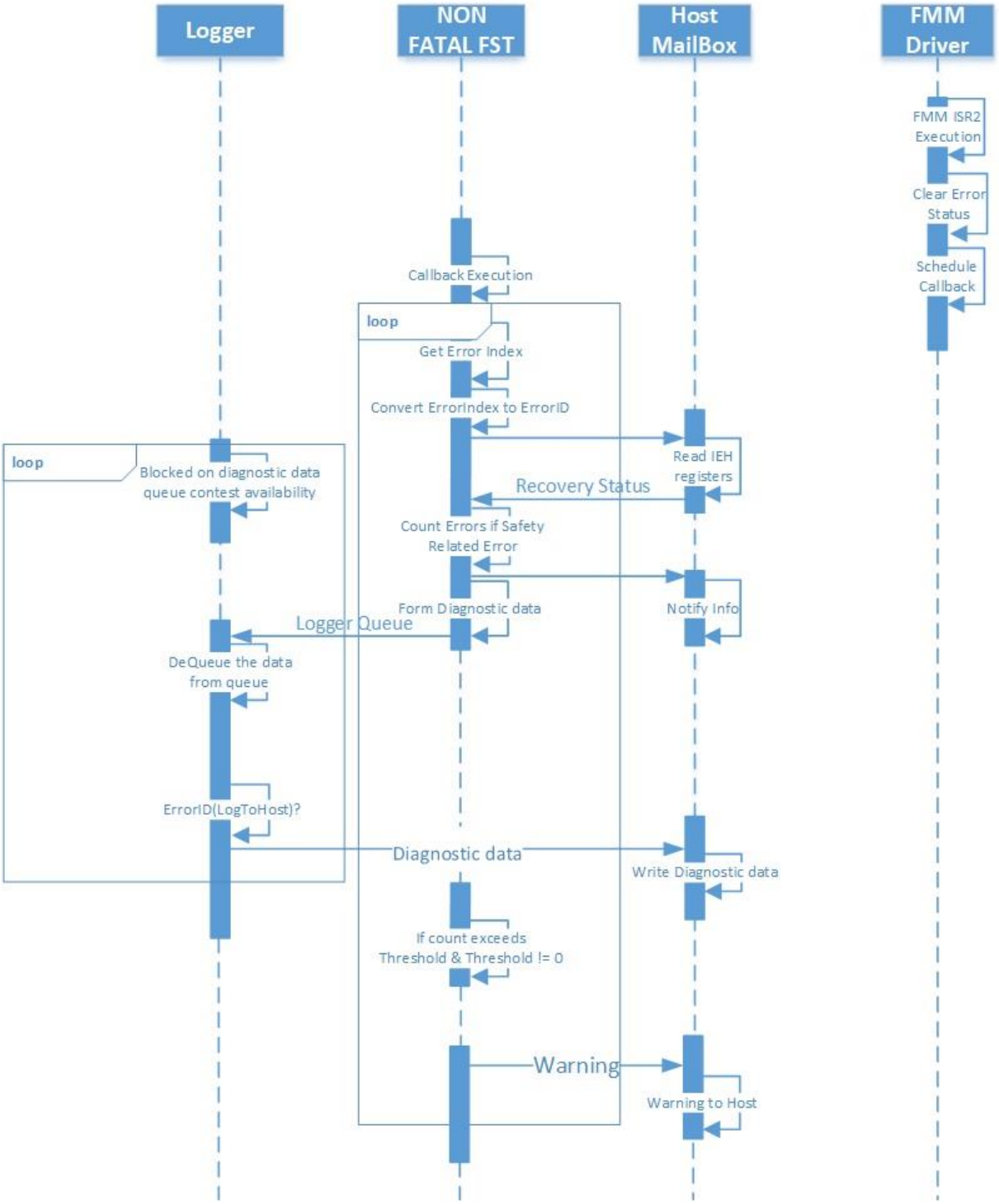
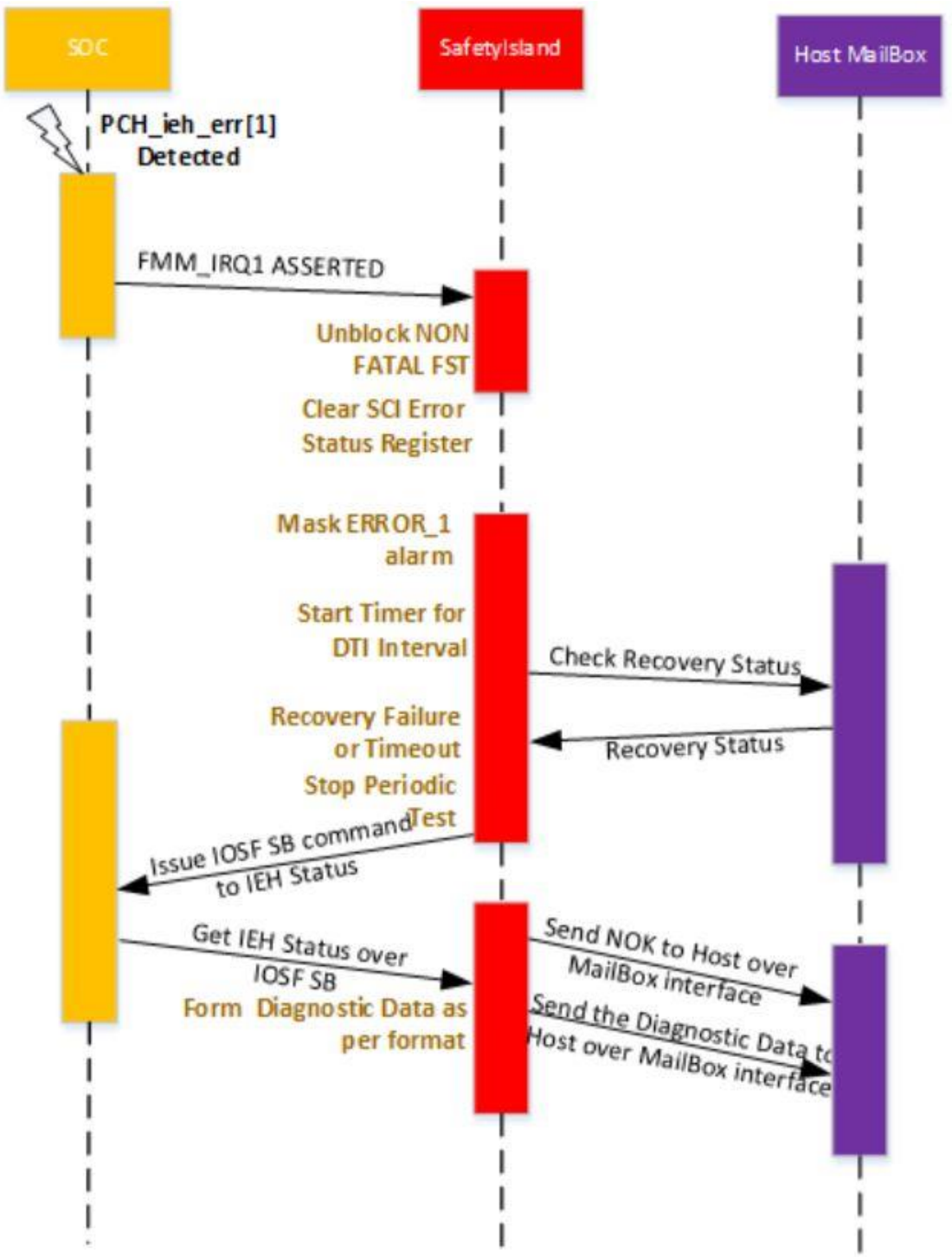


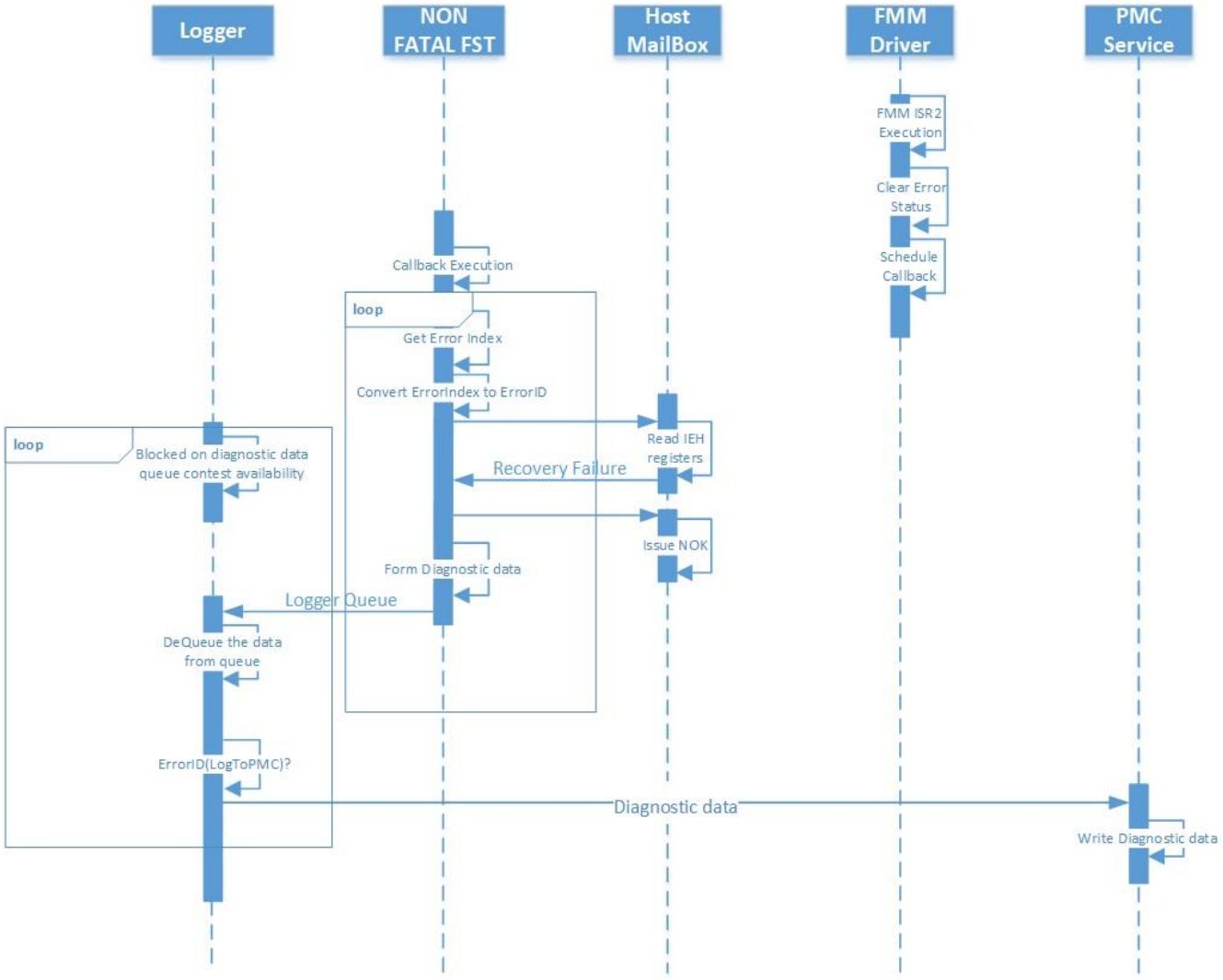
Figure 60 ERROR_1 High Level flow with Recovery Failure



Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590616	Accepted		SIL 3	1. 1oo1 Basic configuration don't have Safety MCU. So Error logs will be provided only to the host		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				2. In response to ERROR_1 ASSERTION, SCI hardware will update the error status register and trigger ISR1 3. The corresponding "error interrupt handler" will clear the status register and schedule a callback. The “interrupt handler” will also pass error number to the callback function. 4. Callback function will unblock the NON FATAL FST, and the FST will query the STL and check if recovery is successful. 5. If ERR_1 recovery is failed, then NON FATAL FST will query the IEH status registers over IOSF and form the log 6. The error log will be transferred to the PMC scratch pad for temporary storage. 7. Next boot host will query SCI and read the error log stored during previous boot.				

Figure 61 ERROR_1 Low Level flow diagram when Recovery Failure



4.7.1.16 1oo1 Smart Configuration IEH ERR1 flow

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850215	Accepted		SIL 3	1oo1 Smart configuration IEH ERR1 flow is same as that of 1oo1 Basic configuration flow, except following differences.1 1. Error Logs will be sent to Safety MCU over SPI 2. If recovery failure, then SCI will assert OKNOK = 01		N/A	Test	Intel Confidential

4.7.1.17 1oo2D Configuration IEH ERR1 flow

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850214	Accepted		SIL 3	1oo2D configuration IEH ERR1 flow is same as that of 1oo1 Basic configuration flow, except following differences. 1. Error Logs will be sent to the other channel 2. If recovery failure, then SCI will assert OKNOK = 01		N/A	Test	Intel Confidential

4.7.1.18 1oo1 Basic Configuration ISI Internal Error

The ISI internal errors are FATAL errors. Each subsystem within ISI has many configuration registers and each of these configuration registers are protected by a parity. Whenever configuration register write happens, parity will be updated and hardware will be continuously monitoring the parity. If there is any SCI internal fatal error SCI will trigger IRQ0. Firmware flow for SCI internal errors is best effort only as, completing the firmware flow requires many SCI sub systems. If there is an error in the SCI subsystem itself then SCI firmware cannot complete its flow.

Figure 62 Internal SCI Error High Level Flow with Error Logging

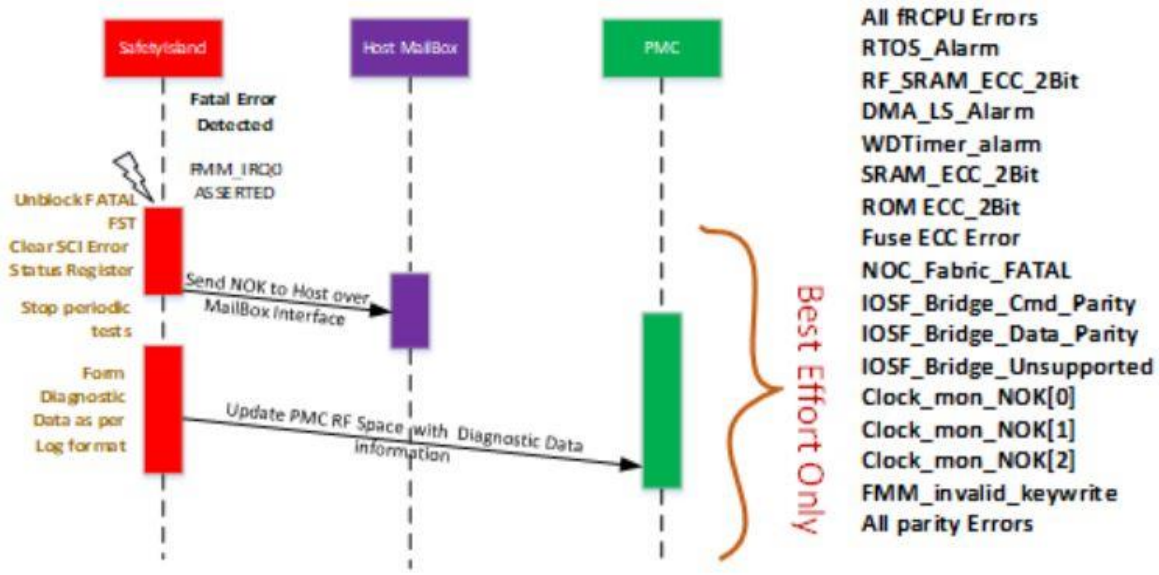
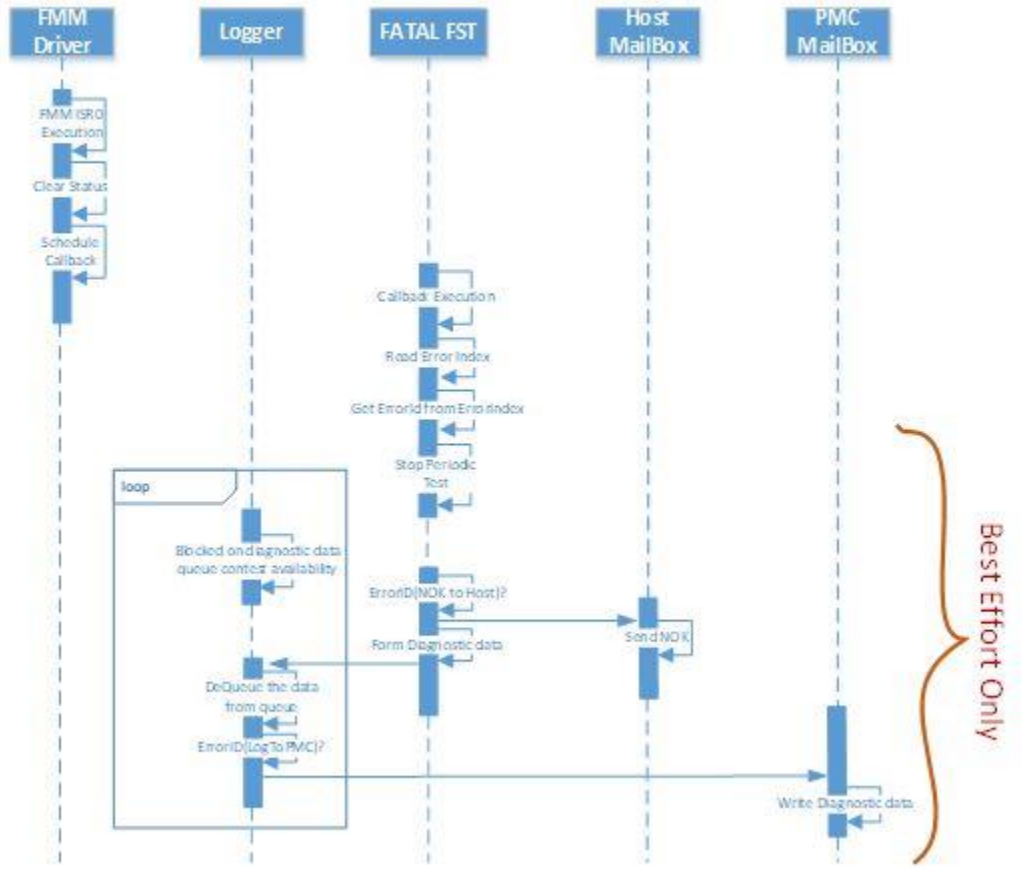


Figure 63 Internal SCI Error Low Level Flow with error log



1. Whenever there is an internal error, FMM will assert IRQ0 and update the status register with the source of the error
2. FMM IRQ0 interrupt handler will clear the status register, schedule a callback and exit
3. 1oo1 Basic configuration don't have Safety MCU. So Error logs will be provided only to the host on next boot
4. Callback function will unblock FATAL FST
5. FATAL FST will issue a NOK to the host
6. FATAL FST will stop all periodic tests
7. Callback function will read the status register and create an error log with error diagnostic information.
8. Callback function will write the error log to PMC scratchpad area. On next reboot, Host software will issue a command to ISI firmware to provide the error log
9. Host software will write the error log information to host nonvolatile area

Note 1: SCI internal error flow is best effort only as there is no guarantee that, processor can execute complete SCI firmware flow under its own error condition

4.7.1.19 1oo1 Smart Configuration Internal Errors with Error Logging

The ISI internal errors are FATAL error flow is same as 1oo1 Basic except the error logs will be forwarded to safety MCU.

4.7.1.20 1oo2D Configuration ISI Internal Parity Errors with Error Logging

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590620	Accepted		SIL 3	In 1oo2D configuration the other channel as MCU. The log will be forwarded to the other channel. Faulted channel will also inform the other channel that it's an SCI internal FATAL error		N/A	Test	Intel Confidential

4.7.1.21 1oo1 Basic Configuration SCI ECC Error

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_850224	Accepted		SIL 3	SCI can detect and correct single bit ECC errors on its SCI SRAM/RF-SRAM/ROM ECC errors. ECC single bit errors are treated as warning and upon threshold the warning condition will be notified to external agents such as host/MCU/other channel. . SCI can detect ECC errors, only when the memory is read/written. Most of the time SCI will be executing periodic tests and hence only same code will be accessed again and again. This may result in an ECC errors in some part of code region never getting detected and they may get converted into ECC double bit FATAL error. To prevent this situation whenever ECC error is detected, SCI firmware perform scrubbing of the code region (write protected region) and detect silent ECC 1-bit errors		N/A	Test	Intel Confidential

Figure 64 SCI ECC 1bit error High Level Flow for 1oo1 Basic Configuration

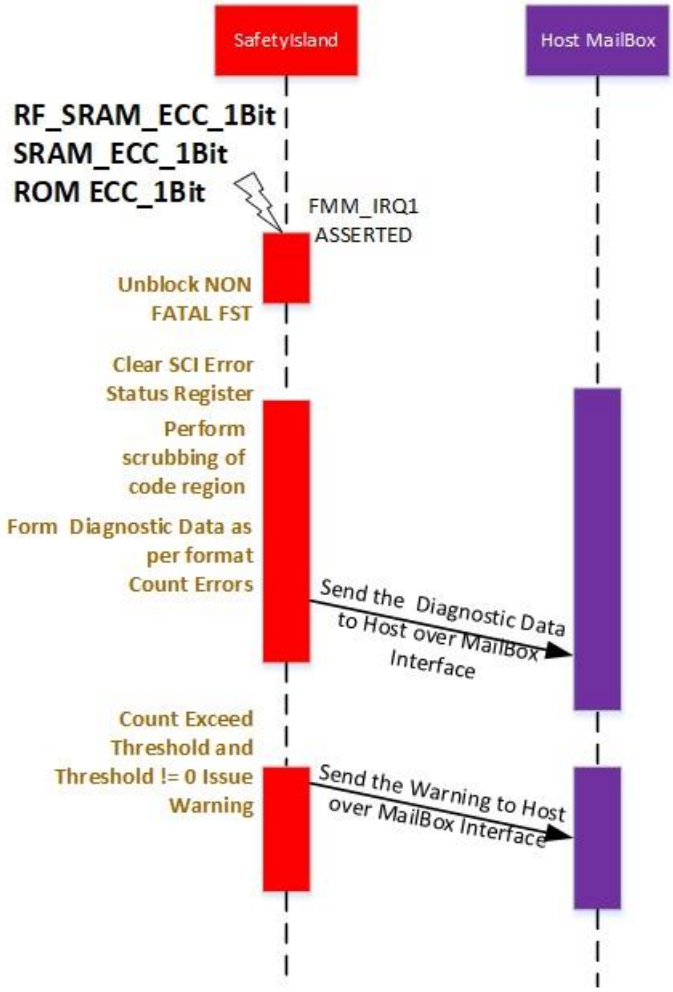
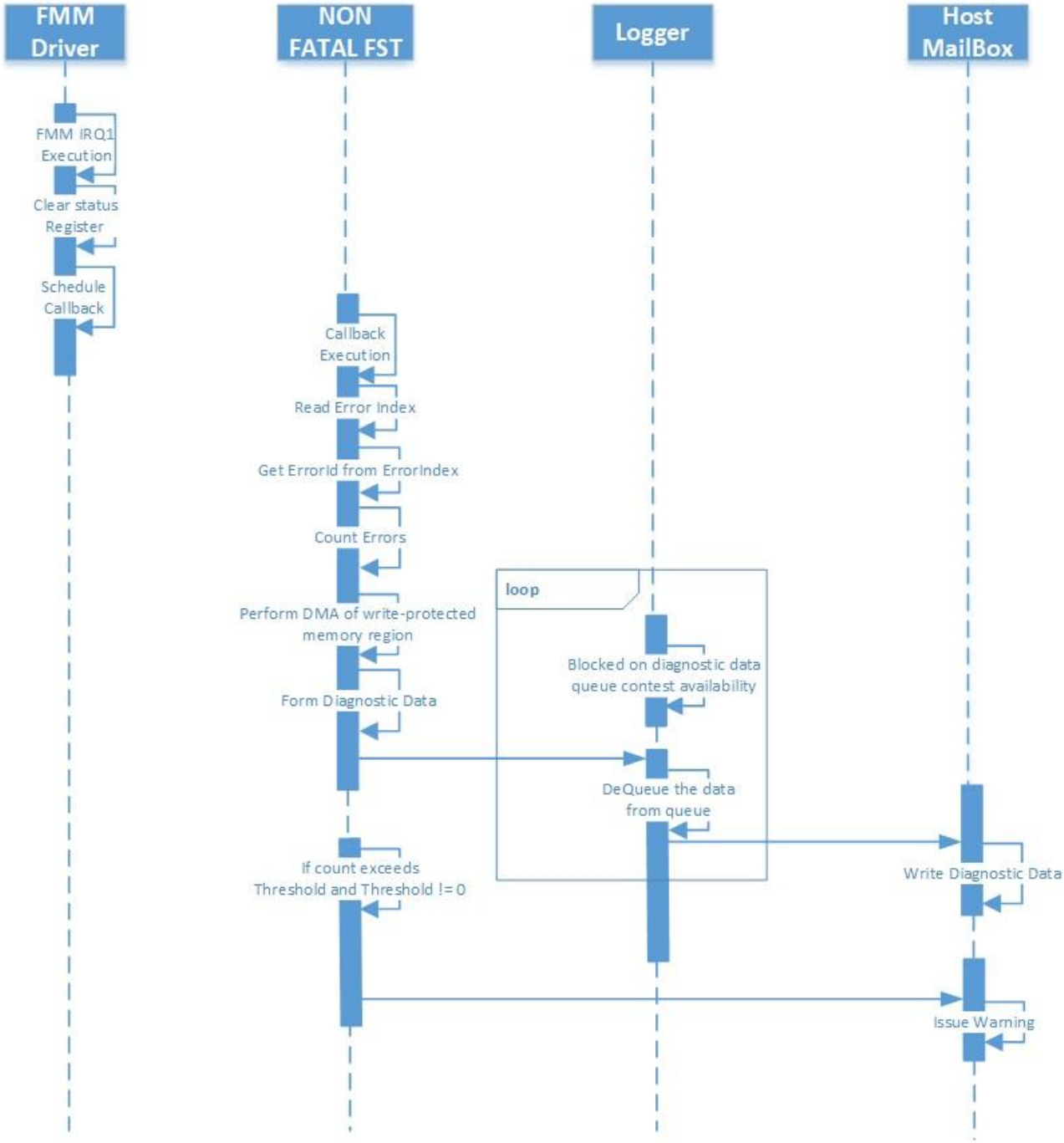


Figure 65 SCI ECC 1bit error Low Level Flow for 1oo1 Basic Configuration



Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850233	Accepted		SIL 3	1. SCI silicon provides a register to		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				configure ECC 1bit threshold value. This register will be configured to 1 2. Every time SCI detects an ECC error, IRQ1 will be asserted. SCI firmware will, mask the SRAM ECC errors, initiate scrubbing of write protected memory region of SRAM (including SRAM and RF) by performing DMA transfer from the write-protected memory region to rw-memory region 3. After the scrubbing, ISI FW will unmask the ECC SRAM errors. 4. The DMA transfer may trigger additional ECC 1-bit detection and these ECC 1-bit errors are counted. 5. A info category message will be delivered to host, if total count is less than threshold. 6. Upon exceeding the threshold, a warning notification will be issued to host.				

4.7.1.22 1oo1 Smart configuration SCI ECC Error

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850236	Accepted		SIL 3	SCI firmware flow for ECC 1bit error is same as that of 1oo1 Basic, however the SCI will issue Alert to safety MCU and also forward the error log to safety MCU.		N/A	Test	Intel Confidential

4.7.1.23 1oo2D configuration SCI ECC Error

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850237	Accepted		SIL 3	SCI firmware flow for ECC 1bit error is same as that of 1oo1 Basic, except following additional flows, 1. Alert notification is sent to the other channel 2. Working channel will query Alert reason from the faulty channel 3. Send the Diagnostic data received to the host		N/A	Test	Intel Confidential

4.7.1.24 1001 Basic firmware detected errors and warning

On detection of an error or warning by a software component in firmware, it will report back the error to upper layer software through error codes. The flow for firmware detected error and firmware detected warning as follows.

Figure 66 Flow for firmware detected errors

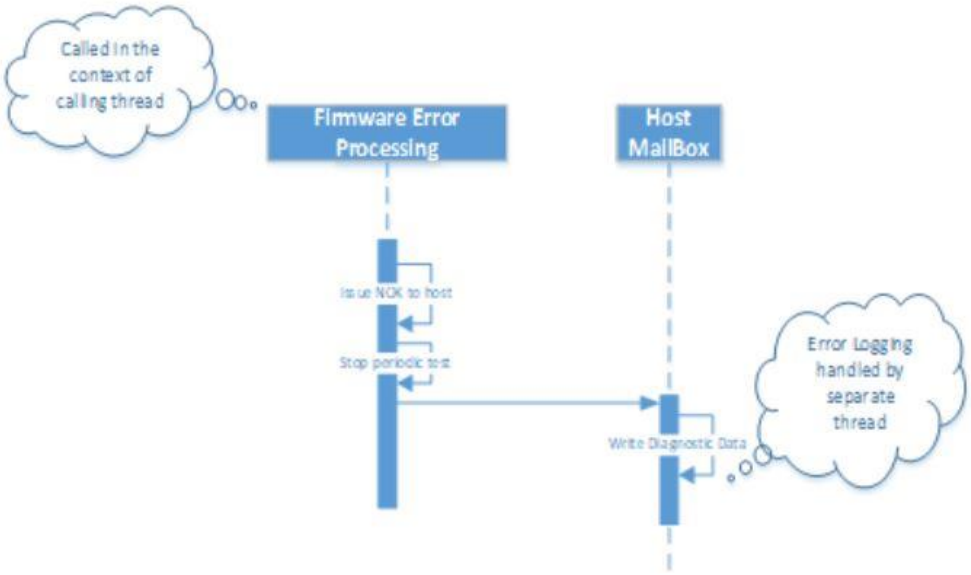
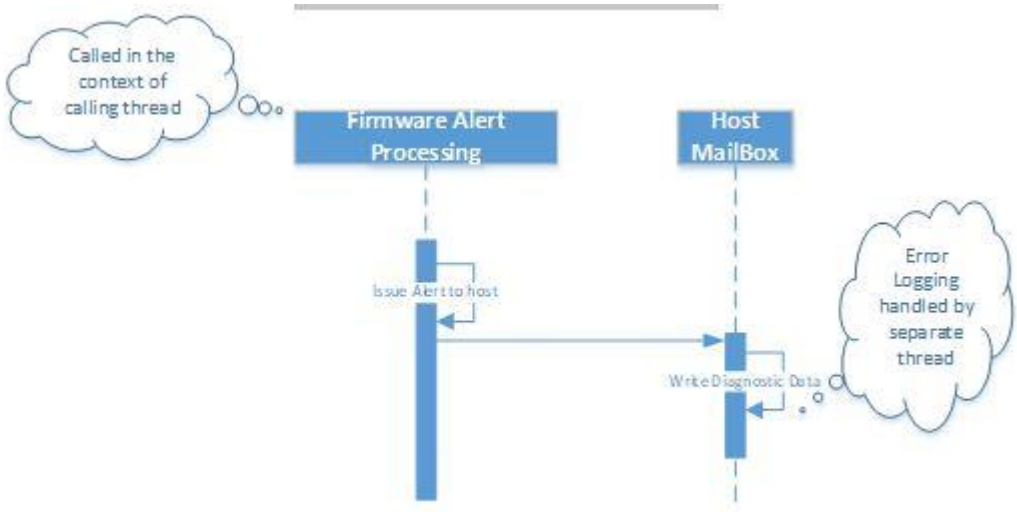


Figure 67 Flow for firmware detected warning



Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850244	Accepted		SIL 3	1. 1oo1 Basic configuration don't have Safety MCU. So NOK and Error logs will be provided only to the host. 2. If the error severity is FATAL, NOK will be sent to host else Alert will be sent to host. 3. The event specific diagnostic data log will come from the firmware itself 4. The diagnostic data log will be forwarded to host		N/A	Test	Intel Confidential

4.7.1.25 1001 Smart firmware detected errors and warning

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 850241	Accepted		SIL 3	SCI firmware flow for firmware detected errors and warning is same as that of 1oo1 Basic, however the SCI will issue Alert to safety MCU and also forward the error log to safety MCU by asserting the ALERT Pin.		N/A	Test	Intel Confidential

4.7.1.26 1002D firmware detected errors and warning

SCI firmware flow for firmware detected errors and warning is same as that of 1oo1 Basic, except following additional flows,

1. NOK reason will be sent to the other channel
2. Error Log will be sent to the other channel

5.0 System Architecture – Elkhart Lake context

This section will describe the overall system architecture for SCI in the context of Elkhart lake platform. This will include:

- Host side SW architecture.
- SCI Boot flows
- SCI security

5.1 Host side SW Architecture

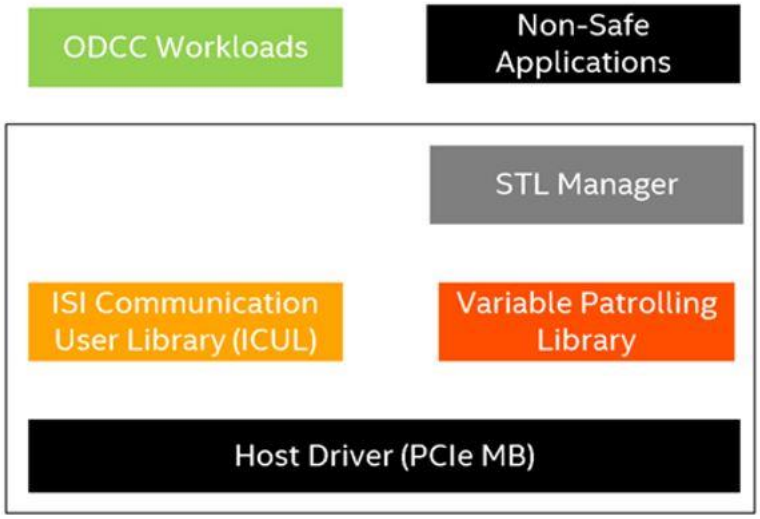
The host side software architecture discussed in this section describes the organization of the major software components running on the EHL CPU. These software components interact with the SCI FW to achieve the required safety goals.

VMM is not yet POR for EHL platform. So all subsections below involving VMM are representative only – for information purpose only. These sections are not used for traceability and development and validation teams will ignore it till VMM becomes POR for EHL platform.

5.1.1 Major components of Host side SW Stack

The high level organization of the host side SW stack is as given below.

Figure 68 Host side SW adaptation on future platforms



1. STL Manager (Referred to as SM): STL Manager is the application that runs STLs provided by Intel. STLs are software test library, which are executed on the EHL CPU to uncover faulty conditions occurring on the CPU. This helps in improving the DC of the entire CPU. STLs are typically run during the platform boot and also at periodic time intervals. The results of STLs are communicated to SCI for taking appropriate actions. The STL tests cover both the core as well as the uncore portions of the EHL CPUs.

The STL Manager assists in following major tasks:

- It schedules the STLs during platform boot as well as periodically.
- The STL manager is also responsible to read the configuration data (PST, DTI values) from the SCI during the platform boot to configure the host stack's behavior.

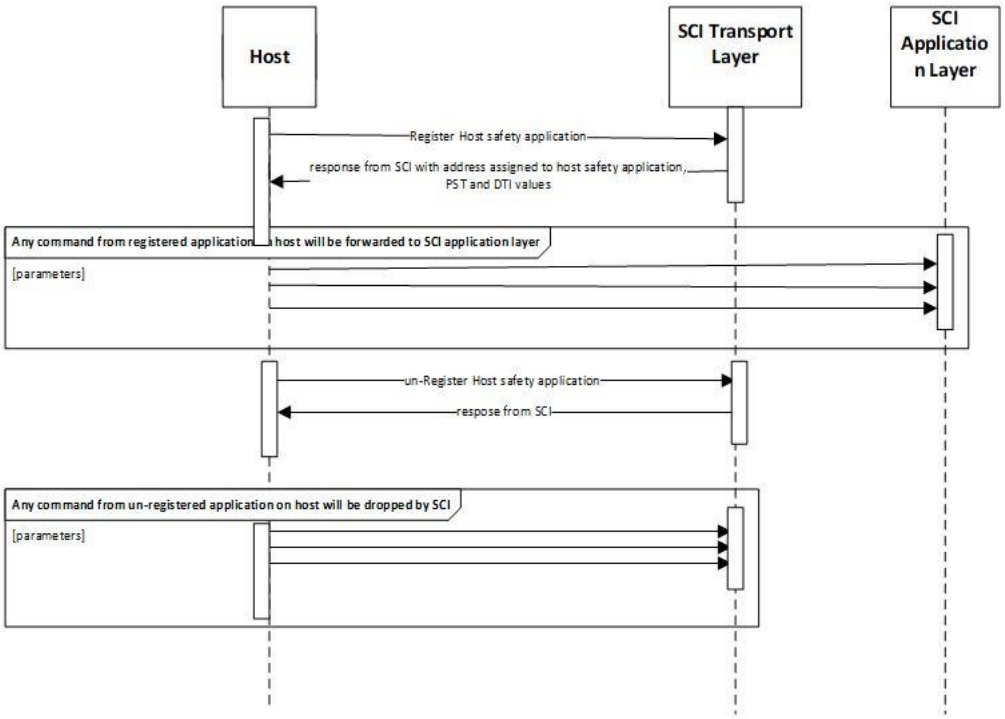
Details on the host side configuration are discussed in Section "Design details of the software components".

2. ODCC Workloads: Safety related workloads (or ODCC workloads) represent the application which perform the intended task to end-customers. Since, these workloads are related to safety, they are developed based on the ODCC framework. Typically, these workloads are spawned on two physically discrete cores and their safety variables are cross-compared periodically by the SCI to ensure their correct behavior.

3. Any workload running on host shall register it with SCI. Only if registered, it can communicate with SCI

STL Manager and ODCC workloads are logical entities for SCI. Depending on customer architecture (host side architecture is owned by customer), it may be same or different application that is ODCC workload and also runs STLs. SCI makes no assumption of STL manager and ODCC workloads to be a single application or different applications.

Figure 69 Registration of safety application with SCI



Non Safety related Workloads: Non safety related workloads are applications which perform tasks not related to the system's safety. These could be executed by the customers to achieve non-safety related goals. These are not discussed any further in this document.

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 971288	Accepted		SIL 3	1. Safety Manager Communication Library (Referred to as SMCL): Safety Manager Communication Library or SMCL is a user		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				space library which provides a high-level programming interface to communicate with the SCI. This library helps in formatting the messages for all outgoing transfers as well as handling the incoming messages from SCI (as per the E2E communication requirements, where in sequence IDs, timeouts, nonce and CRC- are used). The SMCL interacts with the Host side driver entity to achieve this functionality. ODCC workloads and SM are the main entities to interact with the SCI, this SMCL is linked with them. More details on SMCL is provided in following sections. 2. Variable patrolling library: This user space library provides a mechanism to perform variable patrolling. It allows the workloads to evaluate the signature for a list of workload variables (Default is CRC-32 algorithm but this could be overridden by customer's own signature algorithm also) and append the snapshots to accumulated snapshot region. More details on ODCC are provided in following sections. 3. Host driver: The host driver is a standard Linux kernel space driver which performs functionality for interfacing with SCI. It is responsible for managing the hardware interface to the PCIe based mailbox on behalf of all user space components. This includes serialization of all requests, handling interrupts, etc. This driver exposes a device node, which user libraries such as SMCL interface use for transferring messages to and from SCI. It is to be noted that host driver is a quality managed (QM) component while other components such as SM, SMCL and variable patrolling library are safety certified (SC) from Intel. The safety related workloads would be specific to customers and thus would be safety certified by the end-customers.				

5.1.2 Use case scenarios for Host side SW stack

The host software stack is designed to accommodate a number of customer's use case scenarios catering to different safety needs/requirements. These include aspects such as fault-safe Vs fault-operational, use of VMs or not, number of cores/workloads allocated to VMs, number of STL managers, etc.

The high level requirements for the host side SW stack w.r.t. ODCC architecture are as follows,

- Meet the requirements of 1oo1 SIL2, 1oo2D and PLd/Cat3 configurations
- Meet fail safe requirements (EHL) and provide scalability to fail-operational requirements as well
- Scalable architecture with paired channels on different cores, but we can have 0 to multiple cores per channel, 0 to multiple channels per VM

- A single solution designed across different VM configuration and isolation requirements, this means a single solution which would work –

1. No VM support
2. ODCC pair in a single VM
3. All cores in different VMs
4. SR/NSR WLs in different VMs

Customers can extend or come up with their implementation based on one of the above use cases. It is to be noted that in all use cases, multiple workloads can be spawned (using time multiplexing). However it is the responsibility of customer (AoU) to ensure that the workloads running across the cores are able to execute with in the permissible drift.

5.1.3 Design details of the software components

The design details of the individual software components are provided in this section below.

5.1.3.1 Host Driver

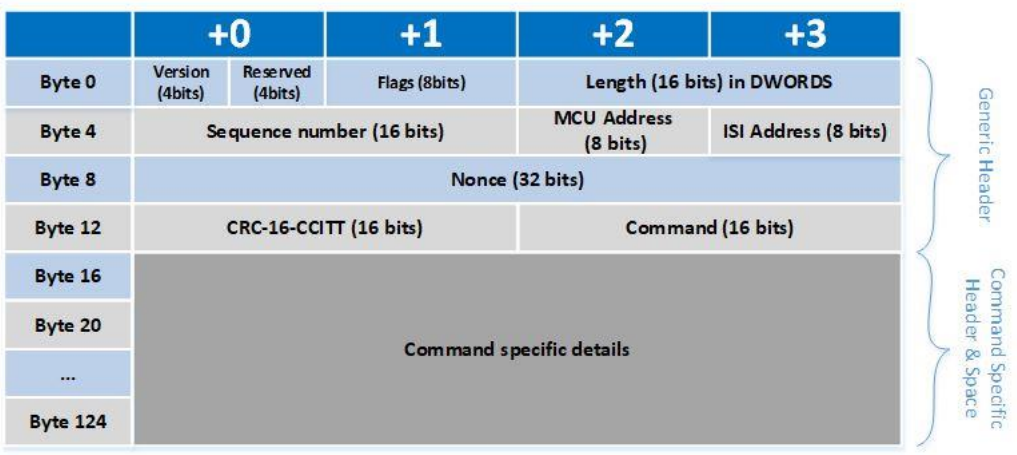
The host driver is a standard Linux kernel space driver which is a QM component. The host driver is based on standard Linux character device driver interface. The QM nature of this module translates to the requirement of keeping this driver simple and performing a pass through operation. The driver thus doesn't contribute to any safety related functionalities such as retry, timeouts and CRC. Those functionalities are thus pushed to user space level and handled by the SMCL.

5.1.3.2 PCIe MB message format

FuSA document ISI Diagnostics interface Specification while the host driver is responsible for programming the PCIe MB hardware to carry out the transmission.

The generic format for a PCIe mailbox message is as given below.

Figure 70 Generic format for PCIe mailbox message



The fields in the generic PCIe mailbox message are given below.

- Hdr (4 bits) – Set the header as 01 always

- Flags (8 bits) – Reserved (4 bits), Response bit, Start of Packet, End of Packet and Retry bit
- Length (16 bits) – Number of DWORDs transferred in this message
- Sequence number (16 bits) – Every packet going out of that PCIe MB is given a sequence number
- Nonce (32 bits) – Snapshot of the SCI timer which acts as nonce
- CRC-16 (16 bits) – CRC16 calculated over entire packet (including padded zeros)
- Host address (8 bits) – Identifies any VMs and/or SMs running in host address space.
- SCI address (8 bits) - Identifies any process running in SCI address space. This is planned to be used for this EHL program.
- Command (16 bits) – Identifies the service requested. Command field is valid only if SOP = 1.

More details on PCIe MB formats are available in a separate FuSA document ISI Diagnostics interface Specification.

5.1.3.3 Safety Manager Communication Library (SMCL)

The SMCL is a user space library which provides an interface for high-level applications such as SM and ODCC workloads to access the SCI through the PCIe MB hardware. The SMCL acts as an intermediary between the applications and host driver. Since the SMCL is a safety certified component, it is delegated with the E2E safe communication responsibilities. It is to be noted that SMCL is statically linked with the SM or ODCC workloads intending to use its services (SMCL is kept private to the process along which it is linked).

OSAL layer: The OS abstraction layer or OSAL provides an abstraction layer for abstracting the OS specific calls. This would help in porting SMCL to other OS such as VxWorks.

5.1.3.3.1 Roles and responsibilities of SMCL

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590691	Accepted		SIL 3	SMCL's main purpose is to provide E2E safe communication for all transfers over the PCIe MB. The roles and responsibilities of SMCL are as follows: • Format the messages to and from SCI. This includes both requests and responses from and to SCI. The formatted messages are then handed over to host driver for transmission as-is. • Adding a sequence number for all outgoing messages and appending the calculated CRC-16. • Evaluate the CRC-16 for all incoming messages and validate the CRC before further consumption. If the CRC comparison fails, it will return error • Provide a way to register and unregister SM and other workloads with the SCI FW. • Perform plausibility check for source and destination, flag bits and version number, sequence number etc. and return an error to the caller if a mismatch is found		N/A	Test	Intel Confidential

5.1.3.4 Variable Patrolling Library

The main purpose of variable patrolling library is to provide a mechanism using which customer applications can perform variable patrolling. Variable patrolling library is a safety certified library which can be linked with the customer's application. By using variable patrolling library, it is possible to upgrade existing applications to ODCC based applications with some modifications. It is to be noted that variable patrolling library is statically linked with the workloads (Note: Variable patrolling library is kept private to the process along which it is linked).

5.1.3.4.1 Components of variable patrolling library

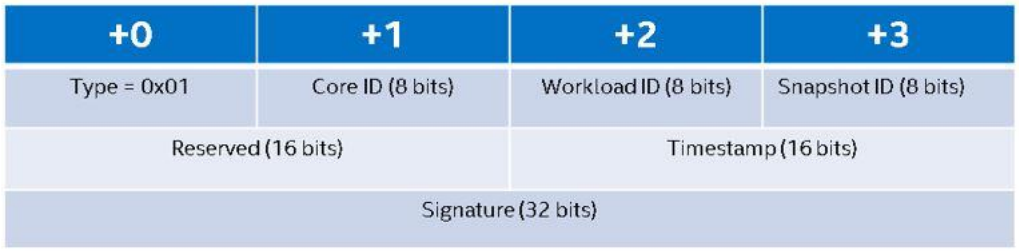
Variable patrolling library generates a snapshot from a list of variables passed in as parameter. The following fields are a part of the generated snapshot:

- Type of snapshot
- Core used to generate the signature
- Workload ID
- Snapshot ID
- Time at which the snapshot was taken
- Signature generated

5.1.3.4.2 API of variable patrolling library

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590699	Accepted		SIL 3	Variable patrolling library provides mechanism to perform variable patrolling. The variables store the program state during the program execution and compared with one another directly. In fact, the signature of these variables can be generated and signatures can be compared instead. This API take in a single variable or a list of variables and then compute the signature on these variables values. It is possible for the customer to override the default signature generation algorithm (which is CRC-32) with their custom implementation. In such a case, the custom signature algorithm would be used in place of CRC-32. The evaluated signature along with details on core which evaluate the signature, the workload ID, the snapshot ID and the time at which the signature was generated along with the evaluated signature forms the snapshot. The format for the snapshot which is logged to the snapshot memory region as provided in the figure below.		N/A	Test	Intel Confidential

Figure 71 Snapshot format for regular variables



A sample API exposed by the variable patrolling library would be as given below. This API takes in an array of variables and generates a single snapshot.

Sample API for regular variable patrolling

```
int variable_patrol_var_list(odcc_context_t *ctxt,
                             void *odcc_var_arr,
                             uint32_t num_elems
                            );
```

5.1.3.4.3 High-level flow for variable patrolling library

The high-level flow for variable patrolling library from a workload perspective is as provided below:

• Workloads calls one of the variable patrolling API either with the regular variables or analog variables. These API are exposed by the variable patrolling library.

• Variable patrolling library increments the snapshot index for that given workload.

- When snapshot is required for regular variables, the variable patrolling library generates the signature of the variables passed in as parameter and then forms the snapshot message.
- When the snapshot is required for analog variables, the variable patrolling library generates the Type 2 snapshot (which has the tolerance number).

• The formulated response is sent to SCI, which then compares the snapshot contents.

5.2 SCI Security

We have following set of assets to protect for SCI:

Table 17 Set of assets to protect for SCI

Asset	Who should have access
Intel Signed FW image and OEM signed configuration data	External Trusted agent who verifies FW (CSE) Authorized DFx access, SCI core
Control Policy Registers	SOC Trusted agents, SCI Core, Authorized DFx

The internal registers are accessible only to SCI core and access controlled by AXI fabric mechanisms.

The IP SRAM usages include:

- Copying FW and Configuration Data from CSE to SCI SRAM
- Data exchange from CSE SRAM

Hence, SCI SRAM should always be secured from any external agents.

SCI IP DMA can also access SRAM for required data transfers from CSE SRAM to the internal SRAM.

Below table lists the Threat Model associated with the SCI Assets:

Table 18 Threat Model

Asset	Protection required	Attack Point	Technique	Mitigation
FW and Config data stored in NVM/UFS	I	Flash contents	Physically modifying the FW image and Config data	We expect SOC to verify FW before loading into IP
FW and Config in CSE SRAM	I	CSE FW	Compromised SW/FW modifies the FW and Config data in CSE SRAM during or after the verification	CSE is within TCB, Expect CSE to protect FW within its IP boundary after verification
FW and Config in CSE SRAM	I	TAP	DFx hooks can modify the SRAM	SOC needs to implement DFx Policy enforcement access to CSE
FW and Config data in SB fabrics	I	TAP	Reconfigure fabrics to re-direct the data after verification	SOC needs to ensure fabrics don't re-direct firmware images. Not possible in SB fabrics that are used to load firmware via CSE
FW and Config data in SCI SRAM	I	DMA	Compromised SW/FW from any other agent in SoC programs SCI DMA to modify SRAM contents	SCI DMA accessible only by ISI CPU and not accessible from HOST
FW and Config meta data in IPC buffers	I	SW	Write into IPC buffers using IOSF fabrics	SAI policies for the IPC buffers allow only trusted agents
FW and Config meta data in	I	TAP	Write into IPC buffers using TAP	SAI policies for the IPC buffers allow

IPC buffers				only trusted agents including DFx agents. TAP tied to DFx Secure Plugin
FW and Config data in SCI SRAM	I	EHL SW	Code injection/flow modification through Mail Box	SCI FW accepts only white listed commands from Mailbox. Commands with known type and payload are acted upon. All other requests are dropped. Also, Host SW cannot directly access SCI SRAM via PCIe MB.
Control Policy Registers	I	Fabric	Compromised SW gets access to Global Control and changes the CP for various Sub-IPs	SAI Based policy enforcement to restrict access to Global Config registers. Only trusted agents allowed.
Control Policy Registers	I	SCI DMA	Compromised SW gets access to SCI DMA to modify the CP in Global Control Registers	Access to Global Config Registers from DMA is restricted. DMA is also access controlled using CPs in Global Config Registers.
Control Policy Registers	I	TAP	Write to Control Policy registers using TAP	TAP is tied to DFx Secure Plugin. Access to CPs allowed only from Trusted SAIs
Control Policy Registers	I	EHL CPU	Write to Control Policy registers	Host CPU SAI's don't get access to Control Policy registers
Debug Logs	NA	NPK	Debug log messages	There are no secrets in SCI FW that may get accidentally exposed through Debug logs. However, SCI

				Production debug traces will be scrubbed through in reviews so that only failure msgs are emitted, without revealing any secret information.

As can be noted in the table above, SCI ROM provides no mitigation to any threat. SCI relies on the CSE FW for image verification and various CPs programmed in the Bridge and the Global Control Registers to secure the SCI Assets.

SCI FW, as an added layer of security will also program the ARM CM3's MPU. The MPU configuration will be such that:

1. Allows only RX permission for SCI FW's code section
2. RW permission for only SCI RW Data section and rest of the available SRAM
3. No access for SCI ROM from SCI FW

5.2.1 SCI Host SW stack (including OS driver)

As per the EHL platform security architecture, the host SW stack is owned by OEMs. Intel will provide only reference code, libraries and drivers that OEM will use as needed. Following this decision, SCI Host SW stack includes the following along with their security status:

1. SCI OS driver: A reference driver that OEM will need to port to their choice of OS. Security owned by OEM
2. Host side libraries: These are static libraries that Intel will provide that includes a SIL SC3 certified code and OS abstraction component. OEM owns the overall usage of these libraries as part of their workloads and they own the security associated with those workloads.
3. Reference code like STL manager etc: OEM will use these components as reference components only.
4. STL libraries: SIL SC3 certified component as static libraries that OEM will use along with their workloads or STL manager. Security owned by OEMs.

5.3 FW Layout and Stitching

SCI FW components are going to be part of the standard IFWI image that for the Elkhart lake platform. Hence, we would use standard MEU tool to create the SCI sub-partition that would be stitched together with the rest of the FW components in the IFWI.

There two SCI FW components:

1. SCI FW image
2. Configuration data

The Configuration data is generated from the Configuration Tool that we will create as per the details mentioned in Configuration Management section above.

As stated in the SCI Security section above, SCI FW image will be signed by Intel and the Configuration Data will be signed by the OEM.

For safety reasons, both FW image and Configuration data will have an associated header that will contain the pre-computed CRC of the respective components. As a part of the signing step, the FW image along with this header will be signed. Same applies to the Configuration data.

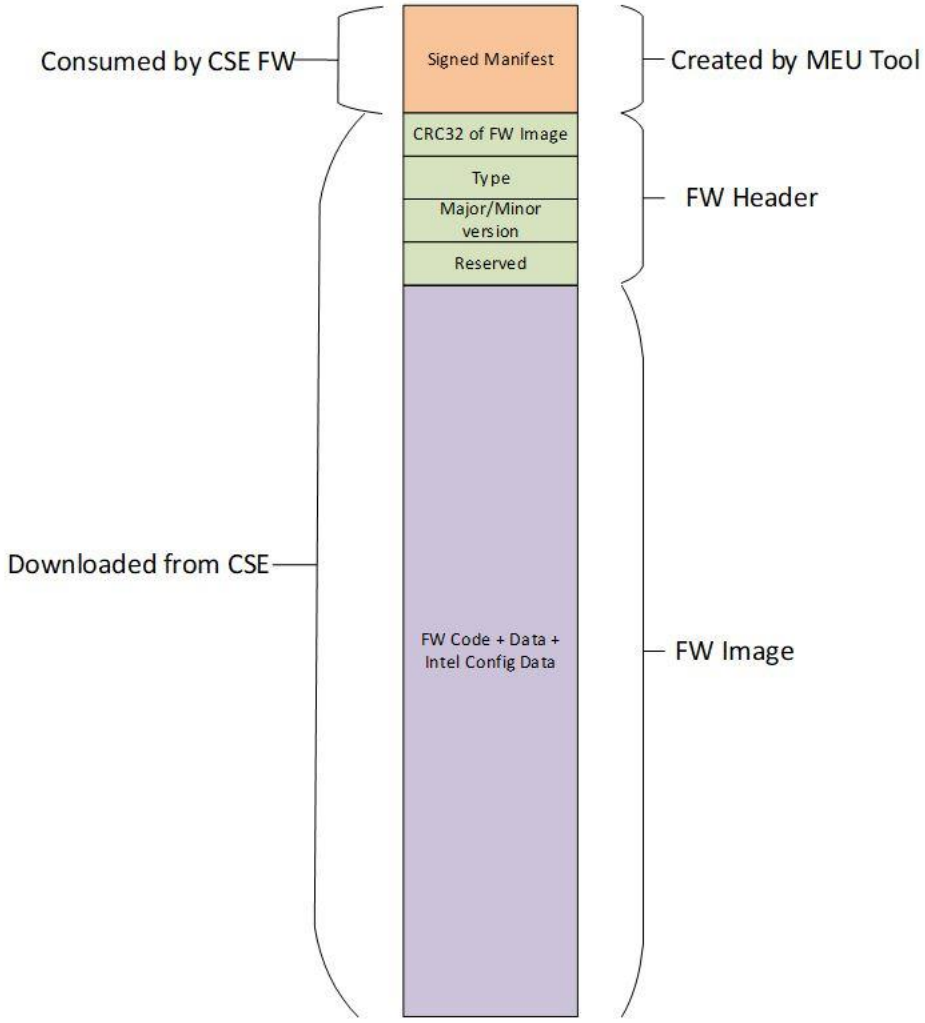
The Header will have the following fields:

Table 19 Header Information

Size in bytes	Component name
4	CRC32
8	Type
1	Major Version
1	Minor Version
18	Reserved

The SCI FW sub-partition will look like this:

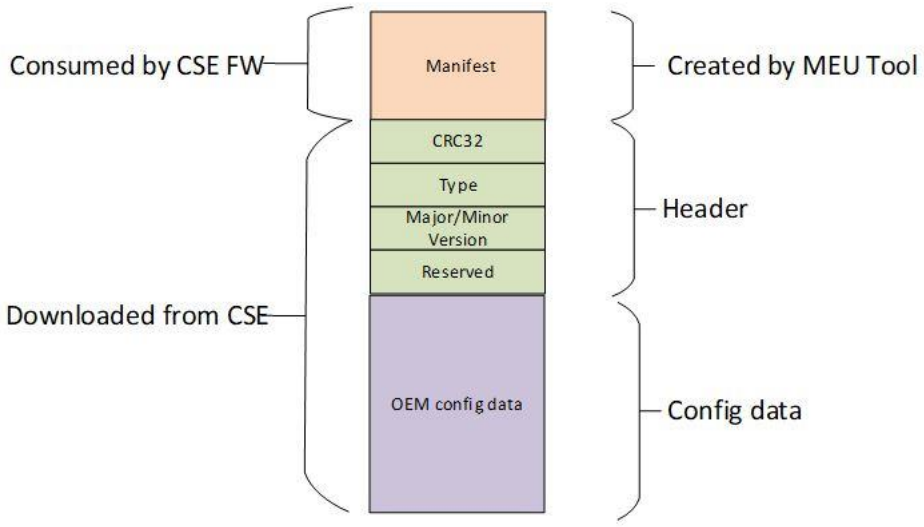
Figure 72 FW layout



To create this layout, SCI FW build scripts will interact with the MEU tool to create the manifest and the signature blob. As stated previously, CSE has limited SRAM to load the SCI FW for verification. Hence, SCI FW build script will take as input the available CSE SRAM size and would divide the FW image into various chunks and may generate multiple manifests.

Similar to FW image, Configuration data too will have its Header and thus in the IFWI SCI config data sub-partition will look like following:

Figure 73 OEM configuration data



5.4 SCI Interaction Points

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590772	Accepted		SIL 3	This section outlines all the commands used by SCI for communication with the various entities in the EHL platform. SCI Interaction Points are detailed below.		N/A	Test	Intel Confidential

Table 20 SCI Interaction Points

Entity	Communication mechanism	Comments
EHL CPU	• PCIe MB for cores • RAVDM for PUnit • GPIO/virtual wires for errors	E2E safe for PUnit and Core
PCH (CSE, PMC and IEH)	• IPC register space for CSE and PMC • SB for IEH, CSE and PMC • GPIO/virtual wires for errors	CSE: redundancy for information PMC: include CRC as part of data
MCU (in case of smart configurations)	• SPI	E2E safe

Other SCI (in case of 1oo2D configurations)	• SPI	E2E safe

5.4.1 EHL – SCI Communication

There are three types of communication used for EHL communication.

1. GPIOs and virtual wires – For communicating events and errors to SCI
2. RAVDM – For PUnit interactions
    1. 1 byte of CRC- added on SCI to PUnit requests and 2 bytes of CRC-added on responses from PUnit to SCI.
    2. Request-response protocol with SCI as master and only one outstanding request at all times.
3. PCIe MB – Used for interactions with host software such as BIOS, Host safety workloads etc.
    1. Two way protocol for exchanging data with CRC-16 protection.
    2. Interrupt on SCI and MSI on EHL side
4. For RAVDM related communication, PUnit in EHL CPU is the only destination endpoint.

5.4.1.1 SCI – PUnit commands

SCI interacts with the PUnit using the RAVDM interface. Following are the list of services provided by the PUnit which SCI uses for various use cases.

- Ping - This service is used by SCI on demand to check the health of SCI2PUNIT mailbox after the initial setup.
- Read PLL Status - This service is used by SCI periodically to check if the CPU PLLs are locked.
- Get domains voltage and temperature - This service is used by SCI periodically to read the CPU's working point voltages for the different voltage rails.
- Read general purpose or MCA bank register - This service is used by SCI to read general purpose registers or MCA bank registers available in uncore region. It is not possible to read the core specific registers using this service. This service is used on-demand for performing error isolation.
- Read ITD (Inverse Temperature Dependency) parameters – This service is used by SCI on demand (typically during boot) to read the fused values in PUnit. During run time these values and an algorithm is used to detect if voltage is not adjusted with temperature variations.
- Get DTS values – This service is used by SCI periodically to read the temperature readings corresponding to different sensors such as core, system agent, DE and graphics sensors.

5.4.1.2 PCIe MB based services

SCI interacts with the various host software such as safety workloads, BIOS using the PCIe MB interface. Following are the list of services provided by host software as well as SCI with respect to this interface.

5.4.1.2.1 To SCI (SCI acts as responder; Host->SCI)

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 590782	Accepted		SIL 3	This category of services in which SCI acts as a responder are classified in to workload specific services and BIOS specific services. **BIOS interactions with SCI** • EHL BIOS will enumerate the SCI PCIe device • BIOS will read the SCI Boot status register from SCI PCIe MB space. If the status value reads as 'BTOK', BIOS will continue rest of the ISI interactions. Otherwise, BIOS will log an error message and continue with the rest of the platform boot. • EHL BIOS sends a PCIe MB message to SCI once it has completed its boot. The SCI FW will have a timer to monitor the host boot. If this message from BIOS doesn't arrive in this configured time limit, SCI FW will generate a NOK signal. **ISI Configuration service** • Configure SCI - To give flexibility to the OEMs to configure various error thresholds and counters in the SCI FW, they can invoke this API to set threshold values for them. Apart from platform configuration and SPI configuration, this API opens up all the config parameters to the OEMs **Workload specific services** (WL acts as requestor/initiator; WL->SCI) • Register Workload – This service is used by the workload executing on Atom to register itself with SCI. In response to this message, SCI assigns address to workload and provides PST and DTI parameters to it. • Share snapshots – This message carries the ODCC snapshots from the host safety WL to the SCI. WL does not expect any response to this message.(Escape Clause: Snapshots not received from host within required time get detected by ODCC FST). • Configure STL – This command is used by the Safety workload to configure various attributes of the STL monitoring in ISI • Configure ODCC – This command is used by the safety workload to • Start Workload – Host safety WL sends this message to mark the		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				start of the workload.. • Stop Workload - Host safety WL sends this message to stop configured safety functions' monitoring.. • Create Timer – This service is used by the workload executing on Atom to request SCI to allocate a timer for WCET and BCET analysis. The allocated timer is identified by an ID. • Configure Timer - This service is used by the workload executing on Atom to request SCI to configure the allocated timer (given by its ID) based on WCET and BCET values. It also allows to configure the action that SCI FW takes when the timer expires • Start Timer - This service is used by the workload executing on Atom to request SCI to start the timer identified with the given ID. • Stop Timer - This service is used by the workload executing on Atom to request SCI to stop the timer identified with the given ID. • Pause Timer - This service is used by the workload executing on Atom to request SCI to pause the timer identified with the given ID. • Resume Timer - This service is used by the workload executing on Atom to request SCI to resume the timer identified with the given ID. • Delete Timer - This service is used by the workload executing on Atom to request SCI to deallocate the timer identified with the given ID. • Get Log Data – This service is used by the workload executing on Atom to request SCI to read and return the logs related as well as the scratchpad contents from AONRF (part of PMC). • Start up STL Results - This service is used by the workload executing on Atom that runs STLs (from Intel provided Library) to notify the SCI on the results of execution of STLs. It also provides host time information. This is used by the SCI FW to sync its local timer with the host timer. This time_of_day time is used for logging.				

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				• Periodic STL Results - This service is used by the workload executing on Atom that runs STLs (from Intel provided Library) to notify the SCI on the results of execution of STLs. There is no response from SCI on this command from workload as SCI expects this command every configured safety loop time. If it is not received every configured time, SCI will assert NOK. • Register for Async Notification - This service is used by the workload executing on Atom to register for Async command notifications from ISI. • Get Async Notification - This service is used by the workload executing on Atom to get async notifications sent by ISI to host.				
SSA_ 849433	Accepted		SIL 3	Start Proof Tests – This service is used by the workload executing on Atom to request SCI to start executing the proof tests.		N/A	Test	Intel Confidential
SSA_ 849434	Accepted		SIL 3	Request SCI NOK generation – This service is used by the workload executing on Atom to request SCI to generate a NOK condition.		N/A	Test	Intel Confidential
SSA_ 849435	Accepted		SIL 3	Request adjacent channel's power cycle - This service is used by the workload executing on Atom to request its SCI to power cycle the adjacent channel through PMIC_EN GPIO.		N/A	Test	Intel Confidential
SSA_ 849436	Accepted		SIL 3	Request SCI Version – This service is used by the workload executing on Atom to request SCI to send its version.		N/A	Test	Intel Confidential
SSA_ 849437	Accepted		SIL 3	Connect/Disconnect adjacent channel relay – SCI shall connect/disconnect adjacent EHL channel based on request from host.		N/A	Test	Intel Confidential

5.4.1.2.2 From SCI (SCI acts as requestor/initiator; SCI->Host)

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 590784	Accepted		SIL 3	This category of services in which SCI acts as a requestor/initiator and WL running on Atom is responder. • Notify NOK and warning – This service is used by SCI to notify the safety workload running on Atom on warnings and/or NOK conditions. SCI does not expect any response to this message. • Escape Clause – This is redundant mechanism to inform host about NOK • Start DTI – This service is used by the SCI to inform its heartbeat to		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				the Host SW stack. This is a broadcast message from SCI and no response is expected by SCI. Host software stack has to ensure that any safety workloads that needs this message receives it. WL can use API provided by SMCL to register with host PCIe MB driver to forward this message to it. • Escape Clause – If a safety workload that has requested this command does not receive this message, it can assume that ISI FW is non-responsive • Request graceful reset – This service is used by SCI to request the host software to perform a graceful restart of platform. This is requested by the slave channel that is undergoing proof test. SCI does not expect any response to this message but starts a timer after sending it. If timer expires and no reset yet, SCI will commando a global reset. • Escape Clause - SCI starts a timer after sending the command. If timer expires and no reset yet, SCI will command a global reset				

5.4.1.3 SCI - External MCU communication

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590786	Accepted		SIL 3	There are two mechanism using which the external MCU communicates with the SCI in case of 1oo1 smart configurations. They are as, 1. 3 wire OK/NOK/ALERT interface using which OK and NOK signals are propagated from SCI to external MCU. 2. FUSA diagnostic SPI interface in which SCI acts as slave and external MCU acts as master. It is to be noted that I2C can be used in place of SPI in certain SKUs. However, for EHL program there is only one FuSA SKU planned and this has SPI as this interface for communication. I2C is hence not applicable for discussions with EHL/SCI. Following are the different messages used in this interface. a. Read ISI Event This command is sent by SMCU to the SCI to query the logs associated with the assertion of NOK event. SCI will de-assert the ALERT Pin if its asserted after receiving diagnostic data from MCU.		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				b. Request Heartbeat – This service is used by safety MCU to request SCI to send its heartbeat. This is used by safety MCU to know that SCI is alive. Asynchronous receive interface thread in FSTM is the consumer of this message. It responds to this command by sending SCI hearbeat. c. Request SCI Version: Safety MCU can use this command to get SCI version.				

5.4.1.4 SCI – SCI communications

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_590788	Accepted		SIL 3	In case of 1oo2D configurations, it is required for the SCI of one channel to communicate with the SCI of neighboring channel. Following are the list of services and messages provided in both the SCIs to allow SCI-SCI communications. • Share snapshots – This message carries the ODCC snapshots from the SCI of one channel to the SCI of another channel. Sender channel does not expect any response to this message. • Escape Clause: Snapshots not received from other SCI within required time get detected by ODCC FST. • Send NOK and warning – This message carries the details on NOK, errors and other warnings from the SCI of one channel to the SCI of another channel. No ack expected for this. • Start Proof Tests – This message is sent by master SCI to request for slave SCI to enter the proof tests execution mode. • Start ODCC – SCI sends this command to other channel on receiving start ODCC from host. The command between 2 SCIs additionally include SCI version number. • Stop ODCC – SCI sends this command to other channel on receiving stop ODCC from host.		N/A	Test	Intel Confidential

5.4.1.5 SCI – PCH communication

There are three types of communication used for PCH communication.

1. GPIOs and virtual wires
    - Error GPIOS to SCI
    - PMC Req-Ack Pins

Virtual wires to CSE for Proof Tests

2. IPC register space
    - For interaction with CSE
    - For interaction with PMC

3. SB interface
    - Used for reading/writing IEH registers.
    - Used for writing to PMC DSW RF.
    - Used for writing to CSE IPC registers

For IPC register space related communication, there are two destination endpoint – CSE and PMC.

5.4.1.5.1 Side Band Interface

Write to DSW buffer (ISI to PMC) – The ISI writes to 128Byte DSW RF using MEMWR operation over IOSF Side-Band. ISI sends the offset in the DSW to be written, size of the data to be written and the actual data as payload in the SB packet.

This mechanism will be used to write Logs to the DSW and also the Proof-Test related information.

Read IEH MCA Error banks for error isolation. The errors are read by the Error FST and are then logged through the SCI Logging interface

5.4.1.5.2 PMC early reset warning/Rest Prep request

These interrupts are for the PMC to send a notification to SCI as an early reset warning or reset prep. SCI needs to do similar handling for both but source of the two from PMC side are different. SCI gets two different interrupts through local control when PMC asserts early reset warning signal or reset prep signal. When SCI gets any of the two interrupts, it terminates all its operations (stops all periodic tests), asserts its OK/NOK signals to 1/1 i.e reset state and asserts the ACK pin to acknowledge PMC that it can go ahead with the Platform reset.

5.4.1.5.3 Virtual wires to CSE

These virtual wires are appear to SCI FW as register bits. SCI can set these bits to request CSE FW to trigger the Proof Test. Rest of the Proof Test flow is similar to TGP proof test flow. Refer section 'ODCC and 1oo2D handling' for further details on Proof Tests.

5.4.1.5.4 CSE IPC commands

Below table lists down the various IPC commands exchanged with CSE for FW and config data pull and for getting the proof test results.

Table 21 IPC commands exchanged with CSE

Direction	Description
CSE --> ISI	ISI FW Loading Query

Direction	Description
ISI->CSE	Response to ISI FW Loading Query
CSE --> ISI	ISI Image Ready
ISI->CSE	Response to ISI Image Ready (e.g. ISI FW Pull completed)
CSE->ISI	Proof Test Result4
ISI->CSE	Proof Test ACK4
ISI->CSE	Generic Failure Status Response

5.4.1.5.5 PMC IPC commands

Platform transition information query (ISI to PMC) – This message is sent by ISI to PMC to query information on last event such as G3 reset, type 7 or 8 reset, validity of buffer, etc.

Read 128 byte buffer contents (ISI to PMC) - This message is sent by ISI to PMC to read the contents of AON buffer given a starting address offset. SEL logs and scratchpad contents which are stored in AON buffer can be thus read using this service.

5.4.2 Handling of Asynchronous commands by SCI

5.4.2.1 One-way commands and it's impact on Black Channel communication

For the communication betweeen the host and the SCI, there are a few commands which dont have a response. Since these commands are periodic in nature, having a response will increase the overall bandwidth.

An analysis was done to check each command to check if there is a safety impact for not having a response message

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_967628	Accepted		SIL 3	All one way commands shall be analysed for potential safety goal violation in case the response is lost due to black channel communication errors.		N/A	Test	Intel Confidential

5.4.2.2 Host Commands

5.4.2.2.1 Host Commands to be handled by interface thread itself

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_849424	Accepted		SIL 3	• Request SCI NOK/alert generation: This will call API from error management. Rest of the flow follows from there. The API includes the host side log and associated SCI handling for the log is similar to other log management scenarios. For alert, in 1oo1S config, ISI will assert ALERT Pin and in case of 1oo2D, it will send the ALERT message to other SCI.		N/A	Test	Intel Confidential
SSA_968659	Accepted		SIL 3	• Request adjacent channel's power cycle : SCI FW will driver the adjacent channel's		N/A	Test	Intel Confidential

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
				PMIC_EN pin to the requested value.				
SSA_ 968660	Accepted		SIL 3	• Request SCI Version • Start Proof Test		N/A	Test	Intel Confidential

5.4.2.2.2 Host commands forwarded by interface thread to other threads/FSTs

1. Share snapshots - forwarded to ODCC FST

2. Create timer, configure, start, stop, pause resume, delete timer - forwarded to timer service thread

3. Start-up STL Results - forwarded to SCI startup tests

4. Periodic STL Results and STL manager Config - forwarded to host STL FST

5. Get Log Data – forwarded to logging service. The logging service will read the logs from PMC storage and send it to the caller. The service will also zero out the PMC DSW space.

5.4.2.2.3 Others

Register Workload - It does not even reach Interface thread. It is handled by transport layer itself

5.4.2.3 Safety MCU Commands

GetISIEvent – Forwarded to Logging service

5.4.2.4 SCI-SCI Commands

1. Share Snapshots – Forwarded to ODCC FST

2. Start ODCC - Forwarded to ODCC FST

3. Stop ODCC - Forwarded to ODCC FST

4. Request SCI NOK/alert generation: This is the command that comes after SCI detects NOK signal assertion. SCI cannot decide anything based on NOK signal detection. It has to wait for this command. So why GPIO interrupt handling? When this command is received, after that pin status can be read, and required flow can be triggered. This will call API from error management. Rest of the flow follows from there. Runs in context of interface thread.

5. Start Proof Test –To be handled by interface thread.

5.5 ODCC and 1oo2D Handling

This topic is explained at the system level in the FST section above.

6.0 Design for Debug

6.1 Survivability Mode

To facilitate debugging of SCI ROM and SCI FW in pre-production environment, SCI will support survivability mode.

Survivability mode is controlled by the fuses for SCI. In addition to the survivability fuse, we will also allocate some fuses which will control disabling some of the run time features of the ROM such as CRC calculation.

In survivability mode, the address 0x0000_0000 is mapped to the beginning of the SRAM. This is where the reset vector of ARM CM3 also gets mapped. Due to this, the SRAM size available in Survivability mode will be 64KB less than the actual SRAM available.

Survivability mode will enable loading the ROM code and runtime FW through TAP directly in the SCI SRAM. Hence, we would not need to interact will the CSE RBE to fetch the runtime FW and configuration data.

Below is the list of some of the configuration fuses that we will reserved for SCI:

Table 22 Fuse allocation

DW	Bits	Fuse bit description
2	31:24	Reserved
	23:20	Debug only Fuses
	19:19	BYPASS_CSE_IPC
	18:18	BYPASS_SRAM_INIT
	17:17	BYPASS_CRC32_CALC
	16:16	Survivability Fuse
	15:14	Reserved
	13:7	ECC bits for 2nd Fuse DW
	6:0	ECC bits for 1st Fuse DW
1	31:0	Reserved
0	31:19	Reserved
	18:1	Function Fuse settings
	0:0	Defines if the SCI has to generate Type7 or Type8 reset to PMC 0 -> Type 8 1 -> Type 7

6.2 Debug Prints

Unique ID	Status	AoU	ASIL	Software Architecture Safety Requirement	Satisfies	Allocation	Verification	Information Classification
SSA_ 849300	Accepted		SIL 3	Debug Prints printed from different places in execution provide debug information. This can be supported both in debug and release builds but not in mission mode in release build. For this, there will be compiler flags DEBUG/RELEASE that can be used for prints intended only in debug release or prints that are also intended for release builds. In addition to these compiler flags, there will be an entry in configuration data that will be used to control/limit providing debug prints and hence use of DTF only in lab use for customer and not in mission mode. An AoU shall be provided to customer to ensure to disable this parameter to avoid debug prints in mission mode. Any debug related interface in SCI firmware shall be blocked through this configuration. SCI uses mipisysT formats for debug prints.		N/A	Test	Intel Confidential

7.0 SCI Firmware Design for Validation

The overall firmware Design for validation involves,

1. Execution of Test Content: All the test content will present in a separate FST called Validation FST.

2. Getting Test results : The test results can be procured through

- Serial logs
- Memory buffer accessible to all FSTs

3. Error Injection and Performance Measurement : This code will be integrated into SCI firmware whenever necessary

To prevent validation hooks interference into normal SCI firmware functionality, all validation related code mentioned above will be placed under VALIDATION_HOOKS. So, along with the DEBUG and RELEASE macros, when VALIDATION_HOOKS macro is enabled, we will get Validation Debug and Validation Release Binaries. Below is the list of architectural requirements to support Firmware Design for Validation

1. Ability to get a log trace on serial console for non-critical data
2. Ability to control the amount and type of log trace on serial console
3. Ability to measure performance of SCI firmware
4. Ability to inject errors into Software flows

Ability to get a log trace on serial console for non-time critical data

The Serial driver will be enabled in validation debug build and validation infrastructure can add the DEBUG messages wherever necessary and collect the necessary information on Serial console. On Validation Release build, Serial logs will be disabled. Note that, this mechanism can be only used for collecting non-critical data

Ability to control the amount and type of log trace on serial console

Serial debug messages provide five log levels of viz. ALL_LOGS, INFO, WARN, ERROR, PERF, NO_LOGS. . Validation engineers can choose appropriate levels based on their requirements. Additionally, validation engineer can decide whether he would like to display all logs or only validation related logs through configuration setting.

Ability to measure performance of SCI firmware

The SCI firmware timer functionality can be used to get the time stamp necessary to measure the performance. The performance results can be saved to a shared SRAM accessible to all FSTs and FSTM. This shared SRAM will be under VALIDTION_HOOK macro.

Ability to inject errors into Software flows

All error injector code will be part of SCI firmware embedded within VALIDATION_HOOKS macro. By default, all error injector will be inactive. Validation FST will activation the error injection by calling the Error injector API. Validation FST can decide, how many times/how long the error injection needs to happen.