

Apollo Lake Platform Intel Architecture Firmware Specification (Volume 2 of 2)

BIOS Specification

For Volume 1 of 2 Refer Doc# 559810

February 2017

Revision 1.2.2

Intel Confidential



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

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

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

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Intel technologies may require enabled hardware, specific software, or services activation. Check with your system manufacturer or retailer.

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

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

Intel® High Definition Audio (Intel® HD Audio): Requires an Intel® HD Audio enabled system. Consult your PC manufacturer for more information. Sound quality will depend on equipment and actual implementation. For more information about Intel HD Audio, Refer [Intel® High Definition Audio](#)

Trusted Platform Module (TPM): The original equipment manufacturer must provide TPM functionality, which requires TPM-supported IA FW. TPM functionality must be initialized and may not be available in all countries.

Code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

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

*Other names and brands may be claimed as the property of others. See Trademarks on intel.com for full list of Intel trademarks.

© 2017 Intel Corporation. All rights reserved.



Contents

1	Introduction	16
1.1	Related Documents	16
1.2	Terminology	17
2	Reset Control	19
2.1	System Reset	19
2.1.1	Cold Reset	19
2.1.2	Warm Reset	19
2.1.3	Generating System Reset from Software	20
2.2	Core Disable	20
2.3	Global Reset	21
2.3.1	PMC_SLP_S3# and PMC_SLP_S4# Stretch	21
2.4	Reporting Reset Mechanism to OS	21
2.4.1	Fixed ACPI Description Table Boot Architecture	22
2.5	Boot [Reset] Timekeeping	23
3	PCI Express* Configuration	24
3.1	PCI Express* Configuration Space Access	24
3.2	PCI Express* Device Mapping	25
3.3	PCI Express* Port Configuration	25
3.3.1	PCI Express* Link Training	26
3.3.2	Initialize Slot Implemented for PCI Express* Port	26
3.3.3	PCI Express* Slot Presence Detection	26
3.3.4	Initialize Slot Power Limit for PCI Express* Port	27
3.3.5	Requirements for Common Clock Mode	27
3.3.6	Link Width Restrictions	28
3.3.7	IA FW GPE Handler for PME Event	28
3.3.8	GPIO Configuration Responsibilities of IA FW	28
3.3.9	Additional APL Specific Configurations	30
3.3.10	RuntimeD3 (RTD3)	31
3.3.11	Function Disable	32
3.3.12	IA FW Configuration of PCIe OBFF	32
3.3.13	Secured Register Lock	32
3.3.14	IA FW Configuration of CLKREQ# Mapping	33
3.3.15	ASPM and Link Capability on the PCI Express* Link	33
3.3.16	Device Type Field of PCI Express* Capability Register	34
3.3.17	VGA Enable Bit	34
3.3.18	Cold Boot/Cold Reset/Warm Reset	34
3.4	Power Optimizer Configuration	35
4	DRAM Configuration	38
4.1	SoC Memory Controller Features	38
4.2	APL LPDDR3 Supported Feature	38
4.3	APL LPDDR4 Supported Features	39
4.4	DDR3L Support	40
4.4.1	Supported DRAM Configurations	41
4.5	Memory Reference Code	41
4.5.1	MRC Flow Selection	41
4.5.2	Platform Programming Considerations	42
4.5.3	MRC Build Flags	42
4.6	BIOS Data Structures	42
4.6.1	Compatible ACPI Table	43
4.6.2	Compatible BIOS Structure Header	46



	4.6.3 BIOS Structure Definitions	46
4.7	System Memory Map	49
5	Internal Graphics Configuration	53
5.1	Internal Graphics Enabling	53
5.2	Graphics Memory Requirements	53
5.3	Dynamic Video Memory Technology	54
5.4	Internal Graphics Disabling	54
5.5	VBIOS/ GOP Driver Post Time Reduction	55
5.6	GT Power Management	55
5.6.1	RC6 Enabling	55
5.7	Display CD Frequency Selection	59
5.8	Multi-Monitor Support	59
5.9	Video BIOS and Video Driver Interface to IA FW	59
5.10	UMA Memory Space Cache-Ability	60
5.11	GOP Driver Implementation	60
5.12	Protected Audio Video Playback (PAVP)	60
5.12.1	PAVP Configuration Steps	61
5.13	Access to MMIO Registers in SMM Mode	61
5.13.1	Memory Map Register Programmed and Locked	61
6	PMC	64
6.1	Configure PCI Function as ACPI Device	64
6.2	PMC I/O BAR	64
6.3	ACPI Space	64
6.4	GCR Space	65
6.5	IPC1	65
6.5.1	IPC1 Messages	66
6.5.2	IA FW Requirement	66
6.5.3	ACPI Requirement	92
6.6	Additional Power Management Programming	92
7	Image Processing Unit (IPU)	93
7.1	Enable/Disable IPU Device	93
7.2	Setup and Lock IMGU Device's Config Space Registers	93
7.3	CSI Lane Configuration	93
7.3.1	FLIS Configuration	93
7.4	GPIO Allocation for IPU	94
7.5	AVStream Virtual Device Enumeration	94
7.5.1	Enumeration of AVStream Virtual Device as Child of GFX	94
7.5.2	Enumeration of AVStream Virtual Device as ACPI Device	96
7.6	ACPI Requirement for IA FW	98
7.6.1	Device Definition	98
7.6.2	Device Specific Method (_DSM)	105
7.6.3	Control Logic ASL Sample Code	111
7.6.4	Camera Module ASL Sample Code	112
7.6.5	Flash ASL Sample Code	118
8	Security Considerations	121
8.1	SoC Security Mechanism for SSA Memory Protection	121
8.2	Isolated Memory Regions (IMRs)	121
8.2.1	IA FW Responsibilities	121
9	Boot Guard 2.0	122
9.1	Architecture	122
9.1.1	Ring Buffer Protocol	122
9.1.2	FIT (Firmware Interface Table)	124



9.1.3	Measured Boot	125
9.1.4	IA FW Requirements	126
9.1.5	BPM Metadata Structure	126
10	Interrupts and IRQs	128
10.1	PCI IRQ Routing on Apollo Lake SoC.....	128
10.1.1	MSI	128
10.1.2	assert_IRQn.....	128
10.1.3	assert_INTA-D	128
10.1.4	SoC Devices Interrupt Routing Capabilities	131
10.1.5	Additional Requirement for IA FW	132
10.1.6	Reporting Interrupt Routing to OS	133
10.2	Processor Interrupt Support	136
10.2.1	Processor Interrupt Delivery Mechanism	137
10.2.2	SoC Settings to Generate NMI	141
10.2.3	NMI Handling in OS Environment.....	142
10.2.4	Routing NMI to SMI	142
10.2.5	SoC Settings to Generate SMI	143
10.2.6	SMI LOCK Bit	144
10.2.7	Setting EOS Bit	144
10.2.8	SMI Status Bits	144
10.2.9	IO Ports 0xB2 / 0xB3	144
10.2.10	Advanced Power Management Control Port (APM_CNT)	144
10.2.11	Advanced Power Management Status Port (APM_STS)	145
10.3	SCI Routing	145
11	UART	146
11.1	UART PCI Addresses.....	146
11.2	UART ACPI Requirement	146
12	xHCI Support	147
12.1	xHCI Controller Options in Reference Code.....	147
12.2	xHCI Controller Initialization	147
12.2.1	Function Disabling xHCI Controller.....	147
12.2.2	Port Disable Override	147
12.2.3	MPHY Initialization	147
12.2.4	Locking xHCI Register Settings.....	148
12.2.5	xHCI Legacy SMI Enabling	148
12.2.6	Additional Settings for xHCI Controller	148
12.2.7	Additional Programming Requirements during USB Initialization.....	150
12.3	Debug Port Support	150
13	USB Device Mode (OTG) Support	151
13.1	Intel® OTG Initialization Flow	151
13.2	IA FW Disabling Intel® OTG.....	151
13.3	Intel® OTG Snoop Programming	151
13.4	Intel® OTG Additional Programming Steps	152
13.5	PCI Mode	152
13.5.1	PCI Mode of Operation	152
13.6	Intel® OTG ACPI Requirements	152
14	Integrated SATA Controllers.....	154
14.1	SATA Controller Initialization	154
14.1.1	Setting SATA Controller Mode	154
14.1.2	Enable Ports	155
14.1.3	Initialize Registers in AHCI Memory-Mapped Space	156
14.1.4	Power Optimizer Configuration	157



14.2	DevSleep Enabling	157
14.2.1	DevSleep Warm Reset Flow Handling	158
14.3	SATA Controller Disabling	158
14.4	Known issues	159
15	Intel® HD Audio.....	160
15.1	Intel® HD Audio Codec Initialization	160
15.1.1	Intel® HD Audio Codec Architecture Introduction	160
15.1.2	Codec Verb Table	162
15.1.3	Codec Initialization Programming Sequence	163
15.1.4	Intel® HD Audio Codec Initialization on S3 Resume	170
15.2	Intel® HD Audio Controller Configuration	170
15.3	Intel® HD Audio PME Event	170
15.4	Intel / iDisplay Audio Link	171
15.5	Additional Intel® HD Audio Programming Steps	171
15.5.1	PCI Mode.....	171
15.5.2	Function Disabling Steps	171
15.5.3	Miscellaneous Initialization Steps.....	171
15.5.4	Static Frequency Switching	171
15.5.5	Registers Lock Down and Related Field.....	173
16	Intel® Smart Sound Technology (Intel® SST)	174
16.1	Intel® Smart Sound Technology Initialization Flow	174
16.2	Intel® Smart Sound Technology Enable/Disable	174
16.3	ACPI Requirements	175
16.3.1	Non-HD-A Link Descriptor Table	175
16.3.2	Feature Support	178
16.3.3	3rd Party IP	178
17	System Management Bus	181
17.1	SMBus Usage Model	181
17.2	Reading a Byte of SMBus EEPROM Data	181
17.3	Block Mode	182
17.3.1	Block Writes.....	182
17.3.2	Block Reads	182
17.4	BIOS / Driver Hardware Semaphore.....	183
17.5	Wake Up from SMBus	183
17.6	SMBus Interaction with TCO	183
17.7	SMBus 2.0 Support	183
17.8	SMBus Block Transfers and TCO	184
17.9	Support for Handling Incomplete SMBus Transactions	184
17.10	SMBus System BIOS Responsibilities	185
17.11	SMI/OS Driver Resource Arbitration	186
17.11.1	SMBus Host Controller Ownership.....	186
17.11.2	SMBus Host Controller Interrupt Ownership	188
17.12	SMBus Handling for System Resets	189
18	High Precision Event Timer (HPET).....	192
18.1	HPET Enabling	192
18.2	High Precision Event Timer Description Table (HPET).....	192
18.3	Interrupt Mapping.....	192
18.4	HPET ASL Code.....	193
19	GPIO AND CFIO Handling.....	194
19.1	GPIO Controller	194
19.2	GPIO and IA FW Dependencies	194
19.2.1	Pad Function Changing	195



19.2.2	GPIO Mode	195
19.2.3	GPIO Interrupt	195
19.2.4	GPIO Wake	196
19.3	GPIO Registers	196
19.3.1	GPIO Memory Space Register.....	196
19.3.2	Community Registers	197
19.4	Serial GPIO Feature	198
19.5	GPIO Interrupt	198
19.6	GPIO Interrupt and Wake Capabilities.....	198
20	LPSS (Low Power Sub System) System	208
20.1	Intel® LPSS Initialization Flow	208
20.2	Disabling Intel LPSS Controllers.....	209
20.3	Intel® LPSS Controller Reset Release.....	210
20.4	Intel® LPSS Power Management Programming.....	210
20.5	Intel® LPSS ACPI Requirements	211
21	Storage Control Cluster	213
21.1	Intel® SCC Initialization Flow.....	213
21.2	Disabling Intel® SCC Devices.....	213
21.3	Intel® SCC Additional Programming Steps.....	213
21.4	Intel® SCC ACPI Requirements	214
22	ISH (Integrated Sensor Hub)	217
22.1	Intel® ISH Initialization Flow	217
22.2	Intel® ISH IMR (Isolated Memory Region) Setup.....	217
22.3	Disabling Intel® ISH	217
22.3.1	PCI Mode of Operation	218
23	SSC (Spread Spectrum Clocking).....	219
23.1	SSC Enable	219
23.2	DDR SSC Registers	219
23.3	HSIO SSC Registers	222
24	Firmware Capsule Update	225
24.1	Firmware Capsule Update (CU)	225
24.1.1	Overall Capsule Update (CU) Requirements	225
24.1.2	IA FW Requirements for Capsule Update via UEFI	226
24.1.3	Pushing the Capsule Update Solution Overview	227
24.1.4	Microsoft Windows* Update Mechanism (for Windows* 8.x and above)	228
24.1.5	OEM Update Mechanism	231
24.2	OS Support	233
24.3	OEM Push Server Recommendations	233
24.3.1	Recommendation of Server-Client Relationship.....	233
24.4	OEM OS Agent Recommendation	233
24.4.1	Agent Communication with OEM Server.....	234
24.5	OEM Push Validation Environment.....	235
24.5.1	OEM Validation Flow Overview.....	235
24.6	IAFW Capsule Update Flow Including Interaction with OS and TXE ...	237
24.6.1	OS and EFI Communication During OTA	239
24.6.2	Pre Update Check Requirements.....	240
24.6.3	Post Update Check Requirements	240
24.7	Firmware Capsule Update Image Layout on Platform.....	240
24.7.1	Boot Partition Descriptor Table	245
24.7.2	Logical Sub-Partition Layout.....	247
24.7.3	Sub-Partition Directory Entry	248



24	24.7.4	SMIP	248	
	24.7.5	Manifest	249	
24.8	Overall IFWI Update Sequence	249		
	24.8.1	Detailed IFWI Prepare for Update Details	253	
	24.8.2	Detailed IFWI Prepare for Update Details (Recovery Flow/Flash Descriptor Override (FDO) strap behaviour)	255	
24.9	IFWI Recovery/Update Module.....	256		
	24.9.1	Unified Recovery/Update Mechanism.....	256	
	24.9.3	Image Authentication.....	257	
	24.9.4	IFWI View from FW Update Perspective	258	
	24.9.5	Boot Partition Table Marking	260	
	24.9.6	Detailed Fault Tolerant IFWI Update.....	260	
	24.9.7	Recovery Mechanism.....	265	
24.10	Firmware Boot Partition Protection	266		
	24.10.1	EMMC Boot Partition PO Protection.....	266	
	24.10.2	UFS Partition PO Protection Feature	267	
	24.10.3	FW Update Types and Use Cases	267	
	24.10.4	Capabilities	267	
24.11	FW Update Requirements	268		
24.12	FW Architecture	268		
	24.12.1	Code Partition Types	268	
	24.12.2	Boot Partition Descriptor Table	269	
	24.12.3	FW Versioning	269	
	24.12.4	Upgrades and Downgrades	270	
	24.12.5	Data Format Sustaining Considerations	271	
	24.12.6	SVN/VCN upgrade/downgrades	271	
	24.12.7	Update_IMAGE_CHECK	271	
	24.12.8	Data Clear	272	
25	TCO	273		
	25.1	TCO IO Space.....	273	
	25.2	Watchdog Operation.....	273	
		25.2.1	Second Timeout STS	273
	25.3	Checking TCO Timer after Reset	274	
	25.4	Additional Settings	274	
26	Legacy Free Systems	275		
	26.1	Legacy Free Systems Handling by IA FW.....	275	
	26.2	Generic Address Structure	275	
	26.3	Debug Port Table	276	
	26.4	Legacy-Free Handling in ACPI Summary	277	
27	Debug Configurations.....	279		
	27.1	Debug MSR	279	
		27.1.1	IA FW Configurations of DEBUG MSR.....	279
	27.2	Intel Processor Trace	279	
		27.2.1	Intel Processor Trace Storage in Memory	279
		27.2.2	Table of Physical Address (ToPA) Scheme	280
		27.2.3	Other System IA FW Requirements for Intel Processor Trace	282
		27.2.4	Intel Processor Trace and SMM	283
	27.3	Intel® Trace Hub (Intel® TH)	284	
		27.3.1	Intel® NorthPeak Initialization Flow	285
		27.3.2	Configure PTI as Output Port if Being Selected	286
		27.3.3	Configure USB as Output Port if Being Selected	286
		27.3.4	USB3 Tracing Enable.....	286
		27.3.5	Post-MRC Initialization Flow	287



27.4	DCI (Direct Connect Interface)	287
27.4.1	DCI Host Enable	287
27.4.2	EXI Power Gating Enable	289
27.5	GOP Debug	289
27.6	Intel Crashlog.....	289
27.6.1	Overview	289
27.6.2	Crashlog SSRAM Organization and Size	291
27.6.3	Description of Architectural Crashlog Components.....	291
27.6.4	Crashlog debugging Tool	292
28	Security Engine	293
28.1	Intel® TXE Overview	293
28.1.1	Intel® TXE Configuration.....	293
28.2	Intel® TXE Platform Requirements	293
28.2.1	Intel® TXE Firmware Disable	293
28.2.2	Intel® TXE Interfaces	294
28.2.3	Intel® TXE – Host Firmware Status Register.....	297
28.2.4	IA FW Boot Paths.....	303
28.2.5	TXE IA FW Requirements	306
28.3	HECI Interface and HECI Message.....	309
28.3.1	TXE Circular Buffer	309
28.3.2	Circular Buffers and Control/Status Registers	310
28.3.3	HECI Interface Initialization and Reset	314
28.3.4	Send/Receiving HECI Message	315
28.3.5	HECI Link-Down Flow	317
28.3.6	Using HECI2 in Windows* OS within SMM.....	318
28.3.7	IA FW HECI Communication Guidance and Timeout	319
28.3.8	Fixed Client Connection Protocol.....	319
28.4	BIOS CSE FW Interaction Overview	321
28.4.1	HECI Protocol.....	321
28.4.2	CSE Interaction Normal Boot Flow	325
28.4.3	Required HECI CSE Syncpoint	327
28.4.4	CSE Interaction DNX	327
28.4.5	Other Configuration Messages.....	329
28.4.6	NVM Access and Handoff	329
28.4.7	Recovery Available MKHI API	329
28.4.8	Power Management	330
28.5	FW Boot Mode	330
28.5.1	Determine Current State	330
28.5.2	Determine Operation Mode	331
28.5.3	Recovery Mode.....	331
28.5.4	Error Mode.....	332
28.6	Firmware Status Definitions (FWSTS DEFS).....	332
28.6.1	HECI1 FWSTS1 DEFS	332
28.6.2	FWSTS2 DEFS.....	339
28.7	Intel® Management Engine Kernel Host Interface (MKHI) Messages .	357
28.7.1	MKHI Header Generic Types and Structures	358
28.7.2	MKHI – Core BIOS Group	358
28.7.3	MKHI – FW Caps Group	358
28.7.4	MKHI – MCA Group	361
28.7.5	MKHI – Secure Boot Group	372
28.7.6	MKHI – DNX Group	378
28.7.7	MKHI – IFWI Update	381
28.7.8	MKHI – General Group	391
28.7.9	MKHI – BUP Common Group	395
28.8	HECI2 NVM Offload	400
28.8.1	READ DATA.....	400
28.8.2	WRITE DATA.....	402



28.8.3	GET FILE SIZE.....	404
28.8.4	LOCK DIR	406
28.8.5	GET PROXY STATE.....	408
28.9	BIOS to ISH Service Messaging	410
28.9.1	SET FILE	410
28.10	Intel® Platform Trusted Technology.....	413
28.10.1	PTT Enabling	413
28.10.2	PTT vs. dTPM Detection.....	413
28.10.3	TPM Commands.....	413
28.10.4	PTT Initialization.....	418
28.10.5	Extending PCRs	419
28.10.6	Enabling/Disable Hierarchies and Establish Random Value for PlatformAuth.....	420
28.10.7	Enabling/Disabling PTT during POST.....	421
28.10.8	Switching between PTT and dTPM1.2 during POST	422
28.10.9	Re-Provisioning	422
28.11	SPI Less EFI Runtime Storage	422
28.11.1	NVM Requirements	423
28.11.2	NVM Access Flow.....	423
28.11.3	NVM under TXE Control – SMM Managing NVM Transactions	426
28.11.4	TXE.....	429
28.11.5	BIOS.....	430
28.11.6	HECI Driver	430
28.11.7	OS Storage Driver	430
28.11.8	ACPI	430
28.11.9	BIOS Performing NVM Transactions via TXE HEIC Driver....	430
28.12	TXE Temp Disable.....	431
28.13	GPP1 Lock.....	431
Appendix A	MONITOR and MWAIT	432
Appendix B	Windows* 8.1 Micro-PEP(µPEP) ASL Support.....	439

Figures

Figure 3-8: BIOS Flow for PCIe* LTR Discovery and Enumeration	37
Figure 5-1. UEFI Boot Mode	60
Figure 8-1. Shared SRAM Layout at CPU Reset	123
Figure 14-1. Intel® HD Audio Codec Node Structure and Addressing	161
Figure 15-1. ACPI Configuration Table.....	176
Figure 24-1. OS and BIOS Level Flow of UEFI Capsule Update	227
Figure 24-2. Windows* Update Firmware Update Package Layout	228
Figure 24-3. Windows* Update Mechanism: Update Package High-Level Flow	229
Figure 24-4. Windows* Update Mechanism: End-User View of Update Package Visibility	230
Figure 24-5. OEM Firmware Update Capsule Layout	231
Figure 24-6. OEM Update Mechanism: Update Package High-Level Flow	232
Figure 24-7. OEM Server Update Mechanism: End-User View of Update Package Visibility..	232
Figure 24-8. OEM Validation Environment Reference Flow.....	235
Figure 24-9. IA FW Workflow for Firmware Update.....	238
Figure 24-10. System Firmware Layout	241
Figure 24-11. Image Layout with Critical Sub-Section to Redundant	242
Figure 24-12. IFWI Prepare for Update Flow.....	250
Figure 24-13. Unified Recovery/Update Mechanism.....	257
Figure 24-14. IFWI View from IFWI Update Perspective	259
Figure 24-15. Fault Tolerant Update with Duplicate Critical Partitions	262



Figure 24-17 Fault Tolerant IFWI Update Single Critical Partition (APL-SPI only)	265
Figure 24-18. Seamless Recovery Flow.....	266
Figure 28-1. Basic Diagram of HECI Interface.....	310
Figure 28-2. Fixed Address HECI Header	320
Figure 28-3. Flow Diagram.....	421
Figure 28-4. Flow Listed below Assumed BIOS Controllable HECI HEAD	429

Tables

Table 2-1. Reset Control Register (I/O Address 0xCF9)	20
Table 2-2. Fixed ACPI Description Table Boot Architecture Flags	22
Table 2-3. Fixed ACPI Description Table Fixed Feature Flags	23
Table 3-1. Apollo Lake Mapping	25
Table 4-1. APL Memory Map Configuration Registers	49
Table 4-2. APL Memory Map Regions.....	50
Table 5-1. CD Frequency Configuration	59
Table 5-2. IA FW Programmed Memory Related Registers.....	61
Table 6-1. IPC1 Message Summary.....	66
Table 6-2. IPC1 ACPI ASL Sample Code.....	92
Table 8-1. FIT Entry Types.....	124
Table 8-2. BPM Metadata Structure.....	126
Table 9-1. IOxAPIC Registers	137
Table 9-2. IOxAPIC Interrupt Routing Table	137
Table 9-3. Interrupt Message Address Format	139
Table 9-4. Interrupt Message Data Format.....	139
Table 9-5. IOxAPIC Input Mapping	139
Table 9-6. SMI_EN Register - bit Definition	143
Table 10-1. UART Devices PCI BAR Register.....	146
Table 10-2. Describes ACPI ASL Code for UART: APL UART ACPI ASL Sample Code.....	146
Table 11-1. MPHY Init Sample Code	148
Table 12-1. Intel® OTG ACPI ASL Sample Code	153
Table 18-1. Per Pad Memory Space Registers Addresses.....	197
Table 20-1. LPSS Block Controllers	208
Table 20-2. Function Disable Programming for Each of the LPSS Controller	209
Table 19-3. Reset Release Programming for Each Intel® LPSS Controller	210
Table 19-4. Clock Gating Programming for Each Intel® LPSS Controller	211
Table 19-5. Sample Code.....	212
Table 20-1. SCC Block Controllers.....	213
Table 20-2. Intel® SCC eMMC ACPI ASL Sample Code	215
Table 24-1 FW Update Requirements	268
Table 24-2. Version Scenarios and Actions	270
Table 25-1. Generic Register Address Structure	275
Table 25-2. Address Space Format.....	276
Table 25-3. Debug Port Table Format	277
Table 26-1. Intel Processor Trace Structure in Memory (Sheet 1 of 2)	281
Table 27-2. EMEACC Pertinent Fields.....	288
Table 27-3. ECTRL Pertinent Fields	288
Table 27-4. EXIPCE Pertinent Fields	289
Table 28-1. HECI1 Initialization	294
Table 28-2. HECI1 Disable	295
Table 27-3. HECI2 Initialization	295
Table 28-4. HECI2 ACPI Mode	296
Table 28-5. HECI2 Disable	296
Table 28-6. HECI3 Initialization	297
Table 28-7. HECI3 Disable	297
Table 28-8. TXE Firmware Status Registers.....	298



Table 28-9. Host Firmware Status Register (HECI1_HFS) Definition (Offset 40h)	298
Table 28-10. General Status Shadow #1 (HECI1_GS_SHDW1) Definition (Offset 48h)	299
Table 28-11. General Status Shadow #2 (HECI1_GS_SHDW2) Definition (Offset 60h)	300
Table 28-12. General Status Shadow #3 (HECI1_GS_SHDW3) Definition (Offset 64h)	300
Table 28-13. General Status Shadow #4 (HECI1_GS_SHDW4) Definition (Offset 68h)	301
Table 28-14. General Status Shadow #5 (HECI1_GS_SHDW5) Definition (Offset 6Ch)	301
Table 28-15. Host Firmware Status Register (HECI1_HFS)– Bits [3:0] Current Working State Values.....	301
Table 28-16. Host Firmware Status Register (HECI1_HFS)– Bits [15:12] Error Code Values.....	302
Table 28-17. Host Firmware Status Register (HECI1_HFS)– Bits [19:16] Operation Mode Values.....	302
Table 28-18. General Status Shadow #1 (HECI1_GS_SHDW1)– Bits [27:16] Extended Status Data	302
Table 28-19. General Status Shadow #1– Bits [31:28] Infrastructure Progress Code Values.....	303
Table 28-20. General Status Shadow #4 (HECI1_GS_SHDW4) – Bits [7:3] Error Status Code (ECS).....	303
Table 28-21. HECL1 MMIO Registers.....	310
Table 28-22. HECL2 MMIO Registers.....	310
Table 28-23. HECL3 MMIO Registers.....	311
Table 28-24. Host CB Write Window (HECI1_H_CB_WW).....	311
Table 28-25. Host Control and Status Register (HECI1_H_CSR).....	311
Table 28-26. TXE Circular Buffer Read Window (HECI1_CSE_CB_RW)	312
Table 28-27. TXE Control and Status Register Host Access (HECI1_CSE_CSR_HA)	313
Table 28-28. DOI3C Control (HECI1_DOI3C).....	313
Table 28-29. Definition for Fixed Address HECL Header.....	320
Table 28-30. Definition for MKHI Header.....	358
Table 27-31. PKTPM Commands	414
Table 28-32. PTT Vendor Capabilities Values	414
Table 28-33. Reason ID in TPM2_GetTestResult	416
Table 28-34. Main TPM Commands	417



Revision History

Document Number	Revision Number	Description	Revision Date
559811	0.5.0	Initial Release	September 2015
559811	0.6.0	<p>PCIE updated in Chapter 3 Firmware Capsule updated in Chapter 24</p> <ul style="list-style-type: none">- The Capsule update methods including server set-up were corrected and simplified.- Mistakes related to TXE were corrected. <p>TXE updated in Chapter 28</p> <ul style="list-style-type: none">- Section 28.4.4.13 Added data clear command HECL message.- Table 28-31 in MKHI_MCA_GROUP_ID, DATA CLEAR command- Section GPP1 Lock added Section 6.5.2, IPC1 CMD update from PMC FW spec 1.0:Added SSC (Spread spectrum) in Chapter 23	November 2015
559811	0.6.1	<p>Global reset update in Section 2.4. Added USB3 DbC Trace enable in Section 27.3.4.</p> <p>Added Camera ACPI requirement in Section 7.6.</p>	December 2015
559811	0.7	<p>Added chapter 9 Boot Guard 2.0 in Vol2 Added BMP metadata to the same section CSE full form in Table 1.2 Updated Chapter 6 for PBASE clarify Added section 4.6 BDAT support Chapter 24 update for SCC Added section 28.5 for GOP debug support Updated section 25.7 for IFWI layout</p>	February 2016
559811	0.7.1	<ul style="list-style-type: none">• Updated Section 2.1.1, 2.4, 2.5 in Chapter 2 and correct CORE_DISABLE_MASK register.• Updated Section 4.5.1 for no training clarify.• Added Section 15.4 for SATA known issue.• Added Section 28.6 for Crashlog.• Updated Chapter 29.4 with below change:	March 2016



Document Number	Revision Number	Description	Revision Date
		<ul style="list-style-type: none">- Overview modified- Softstrap and BIOS enable/disable default setting added.- Added req for OEM to set dTPM location (LPC or SPI).- Modified Failure mode reporting details.- Note on TPM_Startup to fail removed as not applicable to APL.- Removed note on -Before MainTPM commands become available, TPM2_HierarchyControl can only disable Platform Hierarchy - Not applicable.- References to ChangeEPS removed as not recommended to use in BIOS settings.- Added section 29.4 on FWSTS definitions. <p>• Updated Chapter 25 with below change:</p> <ul style="list-style-type: none">- IFWI layout update- Add IA FW Requirements- IFWI Update Module- IFWI Prepare to Update- Data Clear	
559811	0.8	<ul style="list-style-type: none">• Updated all the chapters to align with SIC 0.8.0.• Removed Broxton related sections.• Removed PMC chapter which is maintained in EDS.	May 2016
559811	0.9	<ul style="list-style-type: none">• Updated Chapter 23• Updated Chapter 27• Added Section 18.6• Added Section 2.6	June 2016
559811	1.0	<ul style="list-style-type: none">• Updated Section 23.8• Updated Section 26.6• Updated Figure 23-9 and Figure 23-10• Added Section 27.9	July 2016
559811	1.1	<ul style="list-style-type: none">• Added USB2 L1 W/A in Section 11.2.6	August 2016
559811	1.2	<ul style="list-style-type: none">• Updated Data Clear in Chapter 23 and Chapter 27.• Updated eMMC Configuration for Bx Stepping	October 2016



Document Number	Revision Number	Description	Revision Date
		<ul style="list-style-type: none">• Add PCIE Power Optimizer Configuration Section 3.4• Update PCIE swap mechanism in Section 3.3.11• Added PMC Chapter• Updated USB port diable in Section 12.2.2	
559811	1.2.1	<ul style="list-style-type: none">• Updated Section 27.1.1	December 2016
559811	1.2.2	<ul style="list-style-type: none">• Updated Section 27.4 for DCI Requirement• Corrected Typo in Chapter 3• Updated Chapter 4	December 2016

§



1 Introduction

This document is prepared to assist IA FW writers in supporting the Apollo Lake SoC.

The primary purpose for this document is to supplement the information provided in the External Design Specification for use by IA FW vendors and Intel customers developing their own IA FW. Registers locations are referenced in this document, however, the current external design specification or data sheet should be used along with applicable specification updates for obtaining the current register and bit settings of APL SoC.

This document describes the implementation of a full-featured IA FW. In this document recommendations are made to enable a full-featured IA FW to take advantage of the capabilities of platforms that incorporate the APL SoC chip. This document uses the word "should" to describe this category of features. This document also describes functions that the IA FW must perform in order to ensure correct and reliable operation of the platform. This document uses the word "must" to describe this category of features.

This document will be supplemented from time to time with specification updates. The specification updates contain information relating to the latest programming changes and may also contain resolutions to known errata. Check with your Intel representative for availability of specification updates.

1.1 Related Documents

This document was prepared with the following references. It is assumed that the reader is familiar with these documents.

Document	Document No./Location
Apollo Lake IA FW and BIOS Specification Volume 1 of 2	559810
Apollo Lake SoC SPI and Signed Master Image Profile (SMIP) Programming Guide	559702
Apollo Lake External Design Specification (EDS) Volume 1 of 4	557555
Apollo Lake External Design Specification (EDS) Volume 2 to 4	557556
Apollo Lake External Design Specification (EDS) Volume 3 of 4	557557
Apollo Lake External Design Specification (EDS) Volume 4 of 4	559360



1.2 Terminology

Term	Description
ALT Access Mode	Mode to allow the reading of write-only registers, usually used when saving/restoring register content for power management sleep state implementations.
CFIO	Configurable Input Output
Core Power Well	Main system power, turns off in S3 – S5
Device	A sub-block of the APL SoC, such as USB controller or Low Power Audio engine, designed to handle a specific task, and which will generally be assigned its own address block within MMIO space and be handled by a dedicated driver. In many instances, a group of devices are organized together as multiple functions that together form within a specific "PCI device". Thus, the term "Device" and "Function" are used interchangeably.
CSE	Converged Security Engine
GPIO	General Purpose Input Output
HC	Host Controller
HSIC	High Speed Inter Chip
ISH	Integrated Sensor Hub
LPSS	Low Power Sub System
NC	North Cluster
OTG	On-The-Go
Platform Reset	APL SoC asserts PLTRST# to reset devices that reside on the Primary PCI bus. APL SoC asserts PLTRST# during power-up and when a hard reset sequence is initiated through the CF9h register. PLTRST# is driven inactive a minimum of 1 ms after both PCH_PWROK and VRMPWRGD are driven high. PLTRST# is driven for a minimum of 1 ms when initiated through the CF9h register.
PCI Reset	PCIRST#. This is the secondary PCI Bus reset signal. It is a logical OR of the primary interface PLTRST# signal and the state of the Secondary Bus Reset bit of the Bridge Control register (D30:F0:Reg3Eh[6]).
PCI device PCIe device	A group of up to 8 devices that together share the same PCI bus and device number assignment. They differ according to function number.
Resume Power Well	Trickle from power supply, only turns off when power removed from wall
Resume Reset	Signal that resets the parts of the SoC in the resume power well, generated when the trickle supply turns on.
RTCRESET#	Signal that resets the RTC well (but does not clear the RTC RAM memory contents)
SATA	Serial ATA, an industry specification of the interface for storage controllers and devices.



Term	Description
SC	South Cluster
SCC	"Storage Control Cluster" contains eMMC, SDIO and SDCard controller.
SPI	Serial Peripheral Interface
SSIC	Super Speed Inter Chip
TCO	"Total Cost of Ownership", features that help manage the platform
VLW	Virtual Legacy Wire Message
UMA	Unified Memory Architecture.
HECI	The logical layer of the interface between host and CSE.
HMRFPO	Host ME Region Flash Protection Override. This mechanism originally refers to the ability of IA FW to open up the Intel® Management Engine FLASH region for writing; however this mechanism is also shared with CSE.
IPC	The physical layer of the interface between host and CSE.
MKHI	Intel® Management Engine Kernel Host Interface. A set of host commands for basic configuration and operation of the Intel® Management Engine, which is shared with CSE as well.
EOP	End Of Post
PBA	Pre-Boot Authentication
SHA1	Secure hash algorithm designed by NSA. Hash algorithms compute a fixed-length digital representation known as a message digest of an input data sequence of any length
SMBIOS	System Management BIOS

§



2 Reset Control

This chapter describes platform level resets and how it interacts with IA FW.

2.1 System Reset

A system reset refers to an event where some or the entire system context are returned to its default state. Which sections of the system context are modified depends on the type of reset that occurred. Regardless of the type of reset, the CPU instruction pointer returns to the reset vector at physical address 0xFFFFFFFF0 and CPU starts executing instructions from that location. Since the CPU cannot start program execution from this address in real mode, the first instruction at the reset vector for legacy BIOS is typically a far jump to a location in flash below 1 MB.

The following are the descriptions of the different types of reset.

Note: If devices on the primary bus are disabled, and IA FW needs to re-enable the devices, a global reset is required.

2.1.1 Cold Reset

Known also as Power-OK Based Full Reset or EfiResetCold, this type of reset occurs when the SoC core power well loses power. Triggers for this event include, but not limited to, writing a value of 0xE to the Reset Control Register at IO port 0xCF9, or entering certain system power management states (Suspend to RAM [STR, ACPI G1/S3], Suspend to Disk [STD, ACPI G1/S4], Soft-off [ACPI G2/S5]).

For systems with an embedded controller, PMC_SUSPWRDNACK must be force low before issuing the Global Reset using the 0xCF9 register. Hence it is recommended to ensure that PMC_SUSPWRDNACK is controlled by the PMC and not used as GPIO.

From CPU point of view, this reset type is signaled by the de-assertion of PWRGOOD. All MSRs and general purpose registers within the processor are initialized to their default (power- on) values. Lockable MSRs are unlocked. All cache lines are initialized to the "invalid" state.

2.1.2 Warm Reset

Also known as PLTRST# Reset or EfiResetWarm, this type of reset occurs when the PLTRST# signal is asserted without power state transitions.

From CPU point of view, this reset type is signaled by the assertion of RESET#. Most, but not all, MSRs within the processor are initialized to their default (power- on) values. All general purpose registers are initialized to their default values. Previously locked MSRs remain locked. All cache lines are initialized to the "invalid" state.



2.1.3 Generating System Reset from Software

System resets generated from software are controlled by the Reset Control register at I/O address 0xCF9. This range is always enabled.

Table 2-1. Reset Control Register (I/O Address 0xCF9)

Bit	Description
7:4	Reserved
3	Full Reset (FULL_RST): When this bit is set to 1 and bit 1 is set to 1 (indicating Hard Reset, not Soft Reset), and the RST_CPU bit (bit 2) is written from 0 to 1, APL SoC will do a full reset, including driving SLP_S3#, SLP_S4# active (low) for at least 3 (and no more than 5) seconds. When this bit is set, it also causes the full power cycle (SLP_S3/4# assertion) in response to PMC_RSTBTN#, PMC_CORE_PWROK, and Watchdog timer reset sources.
2	Reset CPU (RST_CPU): This bit will cause either a hard or soft reset to the CPU depending on the state of the SYS_RST bit (bit 1 in this same register). The software will cause the reset by setting bit 2 from a 0 to a 1.
1	System Reset (SYS_RST): This bit determines the type of reset caused via RST_CPU (bit 2 of this register). If SYS_RST is 0 when RST_CPU goes from 0 to 1, then APL SoC will assert the INIT# virtual signal. If SYS_RST is 1 when RST_CPU goes from 0 to 1, then APL SoC will force PMC_PLTRST# active for about 1 ms, however the SLP_S3#, SLP_S4# signals assertion is dependent on the value of the FULL_RST (bit3 of this register).
0	Reserved

2.2 Core Disable

On APL SoC, IA FW can disable Active cores during boot and issue a cold reset for this change to take effect. IA FW can know the available active cores by reading the below register.

CORE_EXISTS_VECTOR MCHBAR MMIO 0x7164			
Bits	Access Type	Default	Description
3	RW	0	CORE3_EXISTS Indication of core.
2	RW	0	CORE2_EXISTS Indication of core.
1	RW	0	CORE1_EXISTS Indication of core.
0	RW	0	CORE0_EXISTS Indication of core.



After verifying the available active cores, IA FW may disable selected cores by writing to the respective bit in the below CORE_DISABLE_MASK register.

CORE_DISABLE_MASK MCHBAR MMIO 0x7168			
Range	Access Type	Default	Description
3	RW	0	CORE3_DISABLE_MASK core3 disable mask
2	RW	0	CORE2_DISABLE_MASK core2 disable mask
1	RW	0	CORE1_DISABLE_MASK core1 disable mask
0	RW	0	CORE0_DISABLE_MASK core0 disable mask

2.3 Global Reset

System will generate a Global Reset for both the Host and TXE when Global Reset is required by TXE, PMC FW Watch Dog Timer (WDT), TXE WDT or a CF9h IO write of 0x06 or 0x0E with "0x0CF9 Global Reset (CF9GR)" bit set, PBASE+0x48[20].

Also, it is highly recommended to issue a Global reset after FWUPDATE.

1. Write the right value to the "CF9h Global Reset" bit (PBASE + 0x48[20]). It is recommended to clear the "CF9h Global Reset" bit prior to loading the OS.
2. Lock the "CF9h Global reset" field by setting CF9LOCK bit (PBASE + 0x48[31]) for production build, but this field should not be locked for manufacturing mode. When the Manufacturing Mode is closed, CF9h Global Reset should be cleared (step#1) and CF9LOCK bit should be set (step#2).

2.3.1 PMC_SLP_S3# and PMC_SLP_S4# Stretch

For Apollo Lake platform, IA FW needs to program the "PMU_SLP_S3# Minimum Assertion Width", "PMU_SLP_S4# Minimum Assertion Width" and "PMU_SLP_S4# Assertion Stretch Enable", etc appropriately in GEN_PMCON_3 register (PBASE + 0x28). Consult with your board designer for optimal settings and refer the relevant "APL SoC External Design Specification (EDS)" for details.

2.4 Reporting Reset Mechanism to OS

The Reset Mechanism is implemented through an 8-bit register described by RESET_REG in the FADT (always accessed through the natural alignment and size described in RESET_REG). To reset the machine, software will write a value (indicated in RESET_VALUE in FADT) to the reset register. The system must reset the computer immediately on the write to this register (that is, software will assume that the processor will not execute beyond the write instruction). The RESET_REG field in the FADT indicates the location of the reset register.



RESET_REG is optional per the ACPI specification and the OS supports it as optional. In the current IA FW, the FADT flag indicating support is not set. Instead the UEFI ResetSystem() is being called for reset

2.4.1 Fixed ACPI Description Table Boot Architecture

This set of flags is used by an operating system to guide the assumptions it can make in initializing hardware on Intel Architecture platforms. These flags are used by an operating system at boot time (before the operating system is capable of providing an operating environment suitable for parsing the ACPI namespace) to determine the code paths to take during boot.

In IA platforms with reduced legacy hardware, the operating system can skip code paths for legacy devices if none are present. For example, if there are no ISA devices, an operating system could skip code that assumes the presence of these devices and their associated resources. These flags are used independently of the ACPI namespace. The presence of other devices should be described in the ACPI namespace as specified in Chapter 6 [of the ACPI specification].

These flags pertain only to IA-PC platforms. On other system architectures, the entire field should be set to 0.

Table 2-2. Fixed ACPI Description Table Boot Architecture Flags

BOOT_ARCH	Bit Length	Bit Offset	Description
LEGACY_DEVICES	1	0	If set, indicates that the system supports end user-visible devices on the LPC or ISA buses. End user-visible devices include devices that have end-user accessible connectors (e.g., COM Port, LPT Port), or devices for which the operating system should load a device driver so that an end-user application can use a device. If clear, the operating system may assume there are no such devices and that all devices in the system can be detected exclusively via industry standard device enumeration mechanisms (including the ACPI namespace). This bit's definition is unassociated with legacy devices connected to an 8042 micro-controller. The presence of an 8042 micro-controller and its associated devices is indicated by the 8042 flag bit.
8042	1	1	If set, indicates that the motherboard contains support for a port 60- and 64-based keyboard controller, usually implemented as an 8042 or equivalent micro-controller.
Reserved	14	3	Must be 0

**Table 2-3. Fixed ACPI Description Table Fixed Feature Flags**

FACP - Flag	Bit Length	Bit Offset	Description
RESET_REG_SUP	1	10	If set, indicates that the system supports system reset through the RESET_REG Register.
Reserved	21	11	Must be 0

2.5

Boot [Reset] Timekeeping

SOC contains an internal timer to calculate the overall SOC boot[Reset] time and relays that information onto CPU Time Stamp Counter (TSC).

SOC starts incrementing the TSC much earlier in the power up sequence. When BIOS reads the TSC @IBBL beginning it will get the overall SOC reset time from Platform reset to IARESET. This time is reported as the overall SOC reset time and is calculated on Sx/G3 boot.

The addition this TSC accumulation would have on Boot/S3 resume time is 140 to 170 mS.

§



3 PCI Express* Configuration

This chapter covers PCI Express* IA FW support on APL SoC. The SoC has a total of three root ports. As part of a PCI Express* Root Complex, there are two RCRBs (Root Complex Register Block), which each consist of a memory-mapped configuration register space region. These RCRBs contain registers including PCI Express*extended capabilities and other implementation specific registers that apply to the Root Complex. Both RCRBs are 4 KB in size.

It should be noted that RCRB space is separate from the standard PCI Express*extended configuration space which employs a Bus:Device:Function number based indexing scheme.

It is assumed that the reader is familiar with PCI Express*specifications and related terminology ("Root Complex", "ASPM", "Virtual Channel", and so on.).

3.1 PCI Express* Configuration Space Access

IA FW must program the PCI Express*Register Range Base Address (PCIEXBAR) by programming the base address of a non-conflicting memory address range into B0.D0.F0.R060h [38:28]. Apollo Lake supports a 256-MB, 128-MB, or 64-MB enhanced configuration space region. If D0.F0.R060h [2:1] = '00b', the region for PCI Express*Enhanced Configuration is 256 MB and Busses 0-255 are decoded. If D0.F0.R060h [2:1] = '01b', the region for PCI Express*Enhanced Configuration is 128 MB and Busses 0-127 are decoded. If D0.F0.R060h [2:1] = '10b', the region for PCI Express*Enhanced Configuration is 64 MB and Busses 0-63 are decoded. The configuration space is mapped above TOLUD within the 2^{39} addressable range. The PCI Express*Register Range Base Address (PCIEXBAR) must be aligned to the selected size boundary.

To enable PCIEXBAR, IA FW must set 'PCIEXBAR Range Enable' bit D0.F0.R060h [0] = '1b'. Once PCIEXBAR is initialized and enabled by IA FW, software can use the PCI Express*Enhanced Configuration Access Mechanism to access the PCI Express*configuration space using BYTE, WORD or DWORD accesses as long as they do not cross a DWORD boundary. This is a valid access mechanism for all system PCI Express. The PCI Express*Enhanced Configuration Access Mechanism is not valid for accessing RCRBs.

The memory-mapped physical address of a given PCI Express*configuration register of a specific Bus:Device:Function can be determined by:

$$\text{PCIEXBAR} + \text{Bus Num} * 100000\text{h} + \text{Device Num} * 8000\text{h} + \text{Function Num} * 1000\text{h} + \text{Register Offset}$$

In a hierarchy that supports the enhanced configuration access mechanism, the first 256 bytes (offsets 0-255) of PCI 2.3 compatible configuration space can be accessed either by using the PCI 2.3 configuration mechanism (OCF8h/OCFCh) or the PCI Express*Enhanced Configuration Access Mechanism. PCI Express*enhanced configuration space (region from offset 256-4095) can be accessed only via the enhanced configuration access mechanism for PCI Express*devices.

For access to the first 256 bytes, IA FW should not mix access types to these registers using both the PCI 2.3 configuration mechanism and the enhanced configuration access mechanism at the same time simultaneously. IA FW should only use one mechanism to access a given register offset at any given time.

3.2 PCI Express* Device Mapping

Table 3-1. Apollo Lake Mapping

PCIe configuration		Dev	Fun	DID	Root Port #	IA FW asl/IA FW POST	Pin Name
X2		20	0	0x5AD6	1	Device(RP01)/Rootport[0]	PCIE_P4_USB 3_P3_TXP/N PCIE_P4_USB 3_P3_RXP/N
		20	1	0x5AD7	2	Device(RP02)/Rootport[1]	PCIE_P5_USB 3_P2_TXP/N PCIE_P5_USB 3_P2_RXP/N
X4	X2	19	0	0x5AD8	3	Device(RP03)/Rootport[2]	PCIE_P0_TXP /N PCIE_P0_RXP /N
		19	1	0x5AD9	4	Device(RP04)/Rootport[3]	PCIE_P1_TXP /N PCIE_P1_RXP /N
	X2	19	2	0x5ADA	5	Device(RP05)/Rootport[4]	PCIE_P2_TXP /N PCIE_P2_RXP /N
		19	3	0x5ADB	6	Device(RP06)/Rootport[5]	PCIE_P3_USB 3_P4_TXP/N PCIE_P3_USB 3_P4_RXP/N

3.3 PCI Express* Port Configuration

This section describes what the IA FW must do to configure the PCI Express* port and contains details for programming the configuration registers.

The basic assumptions are:

- IA FW is able to enumerate and configure all root ports as PCI-PCI bridges in compliance with PCI-to-PCI Bridge Architecture Specification, v1.2.
- IA FW handles/reports the PCI interrupt routing schemes for the root ports and for the secondary buses behind them correctly.



Note: Programming steps for the PCI Express* Port need to be done only if the device is present and enabled. The device presence should be done by checking the Device ID value != 0FFFFh.

3.3.1 PCI Express* Link Training

Link retraining must be triggered by IA FW only after IA FW has completed configuration of link speed.

1. IA FW must read the GPIO configuration from the SMIP to determine which GPIOs are mapped to which PCIe functionality, PERST# and PFET. This is required to know which GPIOs IA FW will be writing to for the following flow.
2. As early in IA FW code as possible (well before enumeration), turn on power to the external devices.
3. 100ms after turning on power to the external devices, IA FW can de-assert PERST# to the device to bring the device out of reset.

Note: REFCLK must be valid/stable at least 100us prior to PERST# de-assertion. This is guaranteed by design, as internal PLL is up and RCOMP for REFCLK IO is complete well before IA FW brings device out of reset through PERST#.

4. Immediately after PERST# de-assertion, IA FW must program BLKDQDA Dx.Fn.R338h [26] and BLKPLLEN Dx.Fn.R0F4h [2] to '0' to allow the link to train.

Note: Steps 2 and 3 should happen within 24ms of each to prevent one side of the link from entering Polling.Compliance.

- Set Dx.Fn.R050h [5] = '1b' to trigger link training.
- Poll Dx.Fn.R052h [11] up to 100ms until it is '0b' to verify that link training is complete.

3.3.2 Initialize Slot Implemented for PCI Express* Port

IA FW must initialize the "Slot Implemented" of the PCI Express* Capabilities Register Dx.Fn.R042h[8] for the PCI Express* port based on the platform implementation. Setting this bit to '1b' will indicate that the PCI Express* link associated with this port is connected to a slot. Setting this bit to '0b' will indicate that the link is connected to an integrated device component or is disabled.

IA FW must assign a chassis-unique number to the Physical Slot Number field of the PCI Express* Slot Capabilities SLCAP register associated with the PCI Express* port (Dx.Fn.R054h [31:19]). This number is not visible to existing legacy OS, but will be used by a PCI Express-aware OS to identify the root port slots.

3.3.3 PCI Express* Slot Presence Detection

IA FW can read the Presence Detect State bit of the SLOTSTS register at Dx.Fn.R05Ah [6] to determine if a card is present. If a card is present, the Present Detect State bit will be set to '1b'.



3.3.4 Initialize Slot Power Limit for PCI Express* Port

IA FW must set the slot power value for the PCI Express* x16 Port to 75 Watts if an x16 device is connected to the port. IA FW must set the slot power value to 10 watts if an x1 device is connected to the port. IA FW must initialize the slot power value if a device is connected to the port even if it is connected without the use of a slot. The slot power consumption/allocation is platform specific. The guidelines are outlined below, based on the PCI Express* Card Electromechanical (CEM) Specification.

IA FW must set the slot power values by initializing the 'Slot Power Limit Value' [14:7] and 'Slot Power Limit Scale' [16:15] fields of the SLCAP register at Dx.Fn.R054h. In order to set a value of 75 W for slot power, IA FW must program '00b' into [16:15] and program 4Bh (75d) into [14:7].

In order to set a value of 10 W for slot power, IA FW must program '00b' into [16:15] and program 0Ah (10d) into [14:7].

The slot power limit initialization must take place before the Option ROM for the downstream device(s) connected to the port is invoked.

Table 3-2. Slot Power Consumption Guidelines

Form Factor/Slot Type	Link Width	
	X1	X16
Default	10W	75W
CEM Standard	10W	75W
CEM Server	25W	25W
CEM low profile	10W	25W
Expresscard*	6.5W	N/A

Note: Intel's CRB BIOS and reference code program 75 W for all enabled PEG controllers. This value may need to be adjusted on a customer platform based on customer platform requirements.

3.3.5 Requirements for Common Clock Mode

IA FW must program the 'Common Clock Configuration' bit of the LCTL Register for both the endpoint and the PCI Express* root port before enabling ASPM. The 'Slot Clock Configuration' [12] bit in the LSTS Register of the device connected to the PCI Express* root port (endpoint device) must be set to '1b', indicating that the device is using the common reference clock. IA FW must accordingly program the endpoint's 'Common Clock Configuration' [6] in the LCTL Register to '1b'. In addition, the IA FW must also program the Common Clock Configuration bit in the LCTL register at Dx.Fn.R050h [6] to '1b'.

Note: A link retrain is required after programming the Common Clock Configuration bit for the change to take effect.



3.3.6 Link Width Restrictions

IA FW must ensure that the PCI Express* link has trained to a link width that matches a configuration supported. Customer's IA FW can use the link training status field to determine what width the PCI Express* link has trained to by reading the LSTS register Dx.Fn.R052h bits[9:4]. A value of 01h denotes that the link has trained to x1, a value of 10h denotes that the link has trained to x16.

Note: If device is failed to train to the recommended link width, customer's IA FW may force the device to x1 and retrain the link in order to display an error message to end user.

3.3.7 IA FW GPE Handler for PME Event

IA FW needs to provide the _GPE._L09() control method in the ACPI name space as the actual handler for the PCI Express* PME SCI event. Once invoked by the ACPI OS, this control method should perform the following:

For each SoC root port:

- Read the PME Status bit (Dx:Fn:0x60 [16]) of the root port. If the bit is clear, then go to step 5.
- Clear the PME Status bit (Dx:Fn:0x60 [16]), by setting it.
- Check the PME Status bit to see if it is set again (in case there was another PME event pending). If it is set, then go back to the second bullet—Clear the PME Status.....
- Clear the PMCS status bit (Dx:Fn:0xDC [31]) by setting it.
- Perform a “Notify(Device, 02)” to notify the OS of the event that occurred at this particular root port location.
- Go to the next root port and repeat steps 1 through 5, until all root ports are done.

3.3.8 GPIO Configuration Responsibilities of IA FW

IA FW must configure the PCIe related platform GPIOs to the function defined in the table below.

PCIe GPIO Function	Bump Name	Muxed Function Name	GPIO/Func#	Notes
PCIE_WAKE0_B	PCIE_WAKE0_B	PCIE_WAKE0_B	1	Bi-directional Open Drain for OBFF support
PCIE_WAKE1_B	PCIE_WAKE1_B	PCIE_WAKE1_B	1	
PCIE_WAKE2_B	PCIE_WAKE2_B	PCIE_WAKE2_B	1	

PCIe GPIO Function	Bump Name	Muxed Function Name	GPIO/Func#	Notes
PCIE_WAKE3_B	PCIE_WAKE3_B	PCIE_WAKE3_B	1	
PCIE_CLKREQ0_B	PCIE_CLKREQ0_B	PCIE_CLKREQ0_B	1 (Default)	Bi-directional IOD, External Pull-Up Required Internal Pull-Up in CFIO can be enabled
PCIE_CLKREQ1_B	PCIE_CLKREQ1_B	PCIE_CLKREQ1_B	1 (Default)	
PCIE_CLKREQ2_B	PCIE_CLKREQ2_B	PCIE_CLKREQ2_B	1 (Default)	
PCIE_CLKREQ3_B	PCIE_CLKREQ3_B	PCIE_CLKREQ3_B	1 (Default)	
PCIE_PERS_T0_B	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	Output Pin, Operates in GPIO mode. Note: The GPIO selected needs to default to 'gpout' and drive 0 so that the PERST# signal is asserted to the device during cold boot, cold reset, and warm reset + RTD3. Alternatively can be 'gpin' with internal pull-down until IA FW is able to configure the pad.
PCIE_PERS_T1_B	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	
PCIE_PERS_T2_B	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	
PCIE_PERS_T3_B	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	
PCIE_PFET0	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	
PCIE_PFET1	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	



PCIe GPIO Function	Bump Name	Muxed Function Name	GPIO/Func#	Notes
PCIE_PFET 2	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	Note: The GPIO selected needs to default to 'gpout' and drive 0 so that the PFET signal is de-asserted to the device during cold boot, cold reset, and warm reset + RTD3.
PCIE_PFET 3	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	Alternatively can be 'gpin' with internal pull-down until IA FW is able to configure the pad.

To configure the GPIO function, IA FW should write to the <pad>_DWO.PMODE register to select the desired function. IA FW must also be careful to correctly set other configurations such as internal pull-up/down per GPIO.

3.3.9 Additional APL Specific Configurations

Because APL supports Sx, the reset configuration for some of the PCIe related GPIOs has changed. For each GPIO described below which uses pltrst_b, IA FW must write to that pads DWO.PADRSTCFG register with a value of 2'11 to select the "soc_RST_B" reset input. This soc_RST_B will be connected to pltrst_b.

Note: WAKE# and CLKREQ# are dedicated GPIOs, but the PERST# and PFET are SOC GPIOs which IA FW must be aware of to write.

Table 3-3. APL PCIe GPIO Reset Configurations

GPIO	Reset
PERST#	pltrst_b
PFET	deep_rst_b
CLKREQ#	pltrst_b
WAKE#	deep_rst_b



3.3.10 RuntimeD3 (RTD3)

The PCIe RTD3 BIOS requirements are covered in two places.

- The main SW/BIOS level flows are documented in the RTD3 PFAS.
- APL specific steps are described below.

3.3.10.1 RTD3 Entry

1. Configure WAKE# to generate SCI (Enable WAKE# GPIO input buffer).
2. Configure PAD_CONFIGURATION_DW0.PMODE = 'b01 (selects native function).
3. PCIE_WAKE0_B = GPIO_205 ==> PAD_CFG_DW0_GPIO_205.
4. PCIE_WAKE1_B = GPIO_206 ==> PAD_CFG_DW0_GPIO_206.
5. Assert PERST# to device.
6. Disable PFET to device.
7. Masks CLKREQPLUS#.

Note: This happens after PERST# assertion so REFCLK to device not lost prior to PERST# assertion

8. Configure PAD_CONFIGURATION_DW0.PMODE = 'b00 (de-selects native function).
9. PCIE_CLKREQ0_B = DGCLKDBG_PMC_1 ==>
PAD_CFG_DW0_DGCLKDBG_PMC_1.
10. PCIE_CLKREQ1_B = DGCLKDBG_PMC_2 ==>
PAD_CFG_DW0_DGCLKDBG_PMC_2.
11. Clear reg BIOS SCRATCHPAD[SOIX_INHIBIT] bit in PMC to allow Soix Entry (from PCIe context).

Note: reg BIOS SCRATCHPAD[SOIX_INHIBIT] is a single bit regardless of the number of root ports.

3.3.10.2 RTD3 Exit

1. Unmask CLKREQPLUS#.
2. Configure PAD_CONFIGURATION_DW0.PMODE = 'b10 (selects native function).
3. PCIE_CLKREQ0_B = DGCLKDBG_PMC_1 ==>
PAD_CFG_DW0_DGCLKDBG_PMC_1.
4. PCIE_CLKREQ1_B = DGCLKDBG_PMC_2 ==>
PAD_CFG_DW0_DGCLKDBG_PMC_2.
5. Disable WAKE# pin to generate an SCI (Disable WAKE# GPIO input buffer).
6. Configure PAD_CONFIGURATION_DW0.PMODE = 'b00 (de-selects WAKE# function).
7. PCIE_WAKE0_B = GPIO_205 ==> PAD_CFG_DW0_GPIO_205.
8. PCIE_WAKE1_B = GPIO_206 ==> PAD_CFG_DW0_GPIO_206.
9. Restore device power by enabling GPIO 100ms min from PFET -> PERST# de-assertion.
10. Release device from reset by de-asserting PERST#.
11. Program L23R2DT, clear BLKDQDA bit in root port to allow link to train.

Note: PERST# de-assertion and BLKDQDA clearing should happen back-to-back to meet 24 ms requirement.



3.3.11 Function Disable

PCIe* function disable is supported through fuses or IA FW configuration. As noted in this section, there are two cases for Function Disable:

- All ports disabled. In this case PMC will leave PCIe in IP-Inaccessible Power gated state.
- One or more ports active. PMC will bring PCIe controller out of reset. PCIe will pull its own fuses to see which ports disabled. Then, IA FW will own the PSF configuration and low power state programming.

Card detection is done 15ms after PLTRST# de-assertion. Prior to this, if IA FW wants to "Function Disable" a port, IA FW should disable the hot plug capability for that port (if enabled).

For each of the PCIe ports that is a candidate for Function Disable, IA FW performs the following:

- Set B0:Dx:Fn:R338h [26] (PCIEALC.BLKDQDA) = 1b
- Poll B0:Dx:Fn:R328h [31:24] (PCIESTS1.LTSMSTATE) until 0xE or 50ms timeout
- Set B0:Dx:Fn:R408h [27] (PHYCTL4.SQDIS) = 1b.
- Function disable the port by setting the related PSF register

NOTE: Should be done through IOSF Sideband programming as the registers might not be accessible if fuse disabled.

If the function 0 is disabled and if there are still other functions existed with the same device number (could be the same or different PCIe* Controller), we need to make the next available function to take up the function 0. For example, root port 1 to be disabled, root port 7 has device connected:

- a. Set "Port '2 Function Number", PCR[SPB] + 0h bit[10:8] to 0 (SPB = PCIe Controller 2) using word access.
- b. Set PCR[PSF_1] + "AGENT_FUNCTION_CONFIG"[RP7] Bit[3:1] to 0
- c. Set PCR[PSF_1] + "AGENT_FUNCTION_CONFIG"[RP1] Bit[3:1] to 6 //what was in that RP7, effectively a swap.
- d. Patch up the related 2 ports 'ACPI device scope entry so that their _ADR reflects their function number correctly

If non ACPI OS is supported, patch up the PIRQ and MP table to reflect the correct interrupt routing information.

3.3.12 IA FW Configuration of PCIe OBFF

Optimized Buffer Flush/Fill (OBFF) is not supported on Apollo Lake. IA FW must set B0:Dx:Fn:064h [19:18] = 00b.

3.3.13 Secured Register Lock

If device 20 is not presented, IA FW should program SRL (secured register lock) bit to make PCD (Port Configuration and Disable) register read only via setting IOSF-SB port 0xB3 offset 0x00 Bit31.



If device 19 is not presented, IA FW should program SRL bit to make PCD register read only via setting IOSF-SB port 0xB4 offset 0x00 Bit31.

3.3.14 IA FW Configuration of CLKREQ# Mapping

As part of the PCIe Subsystem initialization, IA FW must configure the PCIe CLKREQ# mapping to match the SOC/Platform configuration. This is accomplished in the Device Reference Clock Request Mapping registers (DRCRM*). Each individual port mapping is done use the P*CKRQM fields within that register. This programming is required to be done prior to enabling any PM features in the PCIe controller.

Note: Each CLKREQs signal must be associated with the corresponding PCIE_REFCLK to enable the clocks for each port.

Table 3-4. Sample Code for CLKREQs Routing in APL RVP Design

```
// Valid number from 0 to 3, 0xF for port without CLKREQ routed
// for PCIE x2 controller
ScPolicyPpi->PcieConfig.RootPort[0].ClkReqNumber = 2; //RP01
ScPolicyPpi->PcieConfig.RootPort[1].ClkReqNumber = 3; //RP02
// for PCIE x4 controller
ScPolicyPpi->PcieConfig.RootPort[2].ClkReqNumber = 0; //RP03
ScPolicyPpi->PcieConfig.RootPort[3].ClkReqNumber = 0xF; //RP04
ScPolicyPpi->PcieConfig.RootPort[4].ClkReqNumber = 1; //RP05
ScPolicyPpi->PcieConfig.RootPort[5].ClkReqNumber = 0xF; //RP06
```

3.3.15 ASPM and Link Capability on the PCI Express* Link

It is required that L0s/L1 ASPM link states be enabled on the PCI Express* Port and all downstream devices. IA FW should:

- Enable support for L0s and L1 by programming LCAP register
- Active State Link PM support at Dx.Fn.R04Ch [11:10] = '11b'.
- Following the PCI Express* spec, check all the downstream devices ASPM capabilities, calculate acceptance and exit latencies, and then set the supported
- 'ASPM' field of the LCTL register at Dx.Fn.R050h [1:0] accordingly. IA FW should program this field to "11" to enable ASPM L0s and L1.
- Set Dx.Fn.R4Ch [22] to '1b' for PCI Express* 3.0 compliance.
- If any of downstream devices is ASPM incompliance (LCAP [22]), IA FW may check if device violates specification.



- If LCAP [11:10] = '00b' or '10b', disable L0s and L1 for both PCI Express* Port and the downstream device.

Both bytes of this register that contain a portion of the same field must be written simultaneously in order to prevent an intermediate (and undesired) value from ever existing.

3.3.16 Device Type Field of PCI Express* Capability Register

Section 7.8.2 of PCI Express* Specification v1.0a says the following about the Device/Port Type field: "...Extended configuration space capabilities, if implemented on legacy PCI Express* Endpoint devices, may be ignored by software".

However, to ensure an important requirement by the same specification that configuration of advanced features (for example, virtual channel, ASPM, and so on.) must be consistent between both ends of a PCI Express* link, Intel currently recommends that software should not differentiate PCI Express* endpoint devices from PCI Express* legacy endpoint devices when software performs configuration for a link.

Device/Port Type field can and should be used when there is a need. An example of this is when software needs to access a register field that only applies to a root port, but not to an endpoint device.

3.3.17 VGA Enable Bit

When VGA is enabled (VGAEN, Dx.Fn.R03E [4:3]) on the PCI Express* Bridge, the ISAEN bit (R03Eh [2]) of all other peer bridges must also be enabled. This ensures that VGA I/O alias addresses are always directed towards the VGA-enabled interface. If ISAEN bit is disabled in a peer bridge, the VGA alias addresses may instead be directed towards ranges defined by IOBASE/IOLIMIT of peer bridges. This would create a conflict due to two devices claiming the same addresses.

If IA FW erroneously sets the VGA Enable bit, the link will immediately retrain to x1, reducing the available bandwidth for the I/O device. For this reason, IA FW should never set the VGAEN bit in the bridge control register unless it has been determined that a VGA device is present behind the bridge.

3.3.18 Cold Boot/Cold Reset/Warm Reset

In order to allow APL platforms to support a low power boot where a minimal set of platform level devices are on, APL will prevent the PCIe related devices from being powered and initiating link training until IA FW executes. This will prevent those devices from consuming power during cold boot. This puts the responsibility on IA FW to bring up the device and initiate link training.

In order to support this, the PCIe Root Port implements a bit BLKDQDA which holds the link in Detect.Quiet and prevents training. The suggested flow for IA FW is as follows:



1. IA FW must read the GPIO configuration from the SMIP to determine which GPIOs are mapped to which PCIe functionality, PERST# and PFET. This is required to know which GPIOs IA FW will be writing to for the following flow.
2. As early in IA FW code as possible (well before enumeration), turn on power to the external devices through the platform selected PFET GPIOs. This is due to a PCIe Spec defined 100ms delay that must be met before de-asserting PERST# to the device. The earlier in IA FW can be enabled, the more of that 100ms time can be hidden behind other IA FW activities.
3. 100ms after PFET enabled, IA FW can de-assert PERST# to the device to bring the device out of reset.

Note: REFCLK must be valid/stable at least 100us prior to PERST# de-assertion. This is guaranteed by design, as internal PLL is up and RCOMP for REFCLK IO is complete well before IA FW brings device out of reset through PERST#.

4. Immediately after PERST# de-assertion, IA FW must program BLKDQDA Dx.Fn.R338h [26] and BLKPLLEN Dx.Fn.R0F4h [2] to '0' to allow the link to train.
 - a. BLKDQDA holds link in Detect as discussed above.
 - b. BLKPLLEN allows the controller to request the PLL from the MODPHY.

Note: Steps 2 and 3 should happen within 24ms of each to prevent one side of the link from entering Polling.Compliance.

After the above steps have been completed, link training will occur and IA FW can begin enumeration.

3.4

Power Optimizer Configuration

1. Enable support Latency Tolerance Reporting (LTR)
 - a. Based on the flow diagram shown in Figure 3-8, BIOS will determine the override value (snoop and/or non-snoop) to be programmed in B0:Dxx:Fn + 400h
 1. If B0:Dxx:Fn + 400h is programmed, BIOS will also program B0:Dxx:Fn + 404h [1:0] = 11b, to enable these override values
 - Fn refers to the function number of the root port that has a device attached to it.
 - Default override value for B0:Dxx:Fn + 400h should be '880F880Fh'
 - Also set 404h[2] to lock down the configuration
 - Refer to table below for the 404h[3] policy bit behavior.
 2. Program B0:Dxx:Fn + 64h [11] = 1b
 3. Program B0:Dxx:Fn + 68h [10] = 1b
 4. Program B0:Dxx:Fn + D0h [13:12] = 01b
 2. Disable Optimized Buffer Flush/Fill (OBFF)
 - Program B0:Dxx:Fn + 64h [19:18] = 0h
 3. If Endpoint device supports LTR, Device Capabilities 2 Register Offset 24h [11] = 1b,



- a. Program Endpoint LTR Mechanism Enable, Device Control 2 Register Offset 28h [10] = 1b when device supports LTR but is not found in override table (table listing correct latency requirements for devices that supports LTR and also for devices that do not support LTR).
- b. If B0:Dxx:Fn + 400h is not programmed with snoop latency override value, program endpoint max snoop latency register, Latency Tolerance Reporting (LTR) Capability Offset 04h [15:0] = 1003h.
- c. If B0:Dxx:Fn + 400h is not programmed with non-snoop latency override value, program endpoint max non-snoop Latency Register, Latency Tolerance Reporting (LTR) Capability Offset 06h [15:0] = 1003h
- d. If not all devices support LTR
 - o Program PWRMBASE + 3ECh [24, 16] = 0b, 1b

Note: this register should be saved and restored during S3 transitions

Policy bit B0:D28:Fn + 404h[3]

0: The LTR Override values will be in effect when the link is up and in D0, until a valid new LTR

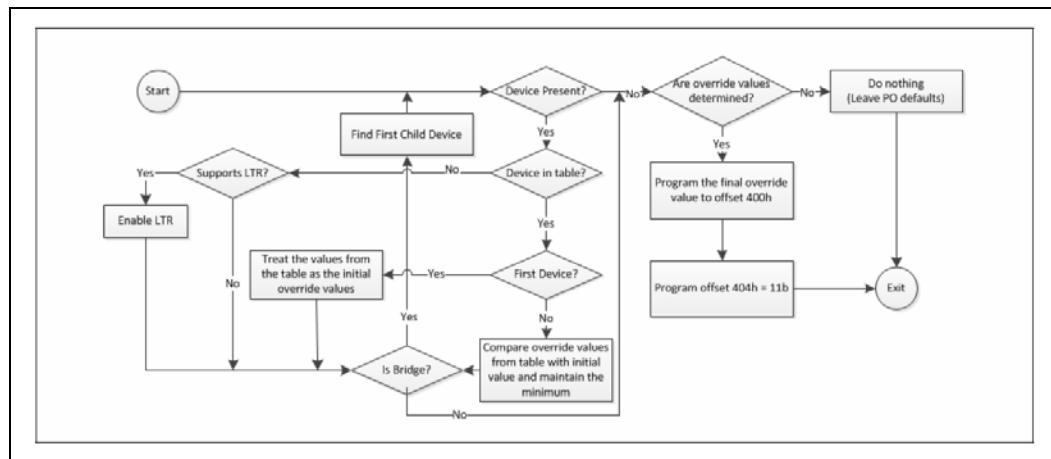
message is received from the device.

1: The LTR Override values will be in effect when the link is up and in D0. LTR values from the

valid new LTR message received from device will be ignored.

404h [3]	404h [0]	404h [1]	400h [15]	400h [31]	Override Policy
0	1	1	1	1	Snoop: Override to values specified by 400h register, until a new LTR message is received. NonSnoop: Override to values specified by 400h register, until a new LTR message is received.
1	1	1	0	1	Snoop: Override to No Requirement, permanently. NonSnoop: Override to values specified by 400h register, permanently
1	1	1	1	0	Snoop: Override to values specified by 400h register, permanently. NonSnoop: Override to No Requirement, permanently.
1	1	1	1	1	Snoop: Override to values specified by 400h register, permanently. NonSnoop: Override to values specified by 400h register, permanently.

Figure 3-1: BIOS Flow for PCIe* LTR Discovery and Enumeration



§ §



4 DRAM Configuration

Note: All of the IA FW requirements specified in this section are included in the SoC Memory Reference Code.

4.1 SoC Memory Controller Features

Features listed below are supported by the memory controller. The memory configuration supported by each relevant project might be different.

Technology	Features	APL
LPDDR3	Data Rate	Up to 1866
LPDDR3	Device Density	4Gb, 6Gb, 8Gb, 12Gb, 16Gb
LPDDR4	Data Rate	Up to 3200 for APL
LPDDR4	Device Density	8Gb, 12Gb, 16Gb
DDR3L	Data Rate	Up to 1866
DDR3L	Device Density	4Gb, 8Gb

4.2 APL LPDDR3 Supported Feature

Features	APL
Connection Technology	BGA
Data Rate (MT/s)	1066, 1333, 1600, 1867
DRAM Device Data Width	2 x16, x32
DRAM Device Density	4Gb, 6Gb, 8Gb, 12Gb, 16Gb
Banks / DRAM Chip	8
Burst Length	2 x 8
Burst Type	Sequential – 000 and 100 only
Read Latency	8 - 16
Write Latency	3 - 16
Auto Precharge Read/Write	No
Refresh Rate change based on DRAM temperature	Yes
Timing Derating based on DRAM temperature	Yes

Features	APL
Refresh Burst Window (tREFBW)	No
Per Bank Refresh	No
Self-Refresh	Yes
Partial Array Self Refresh (PASR)	Yes
Power Down	Yes (PPD Only)
Deep Power Down (data loss)	No
ZQ Calibration	Yes
Mode Register Access	Write and Read
CK/CKB Clock Stop	Yes
CK/CKB Clock Tristate	Yes
ODT (Command and Always-on for Write Leveling)	Yes
CA Training	Yes
tCPDED	Minimum 2 DES after PD/SR entry
Write Leveling	Yes

4.3 APL LPDDR4 Supported Features

Features	APL
Connection Technology	CoPoP, BGA
Data Rate (MT/s)	APL: 8Gb, 12Gb, 16Gb
DRAM Device Data Width	x16, 2 x16 (BGA)
DRAM Device Density	4Gb, 6Gb, 8Gb, 12Gb, 16Gb
Banks / Channel	8
Burst Length	16 (BGA configs), 32 (for 32B and 64B PMI Reads)
Burst Type	Sequential – (Critical 32B First)
Read Latency	10 - 40
Write Latency	8 - 34
Auto Precharge Read/Write	No
Refresh Rate change based on DRAM temperature	Yes
Timing Derating based on DRAM temperature	Yes
Per Bank Refresh	No
Self-Refresh	Yes



Features	APL
Partial Array Self Refresh (PASR)	Yes
Power Down	Yes (PPD Only)
ZQ Calibration	Yes
Mode Register Access	Write and Read
CK/CKB Clock Stop	Yes
CK/CKB Clock Tristate	Yes
ODT (Command and Always-on for Write Leveling)	Yes (No for low power freq)
CA Training	Yes
tCPDED	7.5 ns
Write Leveling	Yes
Data Bit Inversion	No
Post Package Repair	Yes (Performed by CPGC)
Periodic Training (DQS to DQ)	Yes
Self-Refresh Abort	Yes (MR and tXSR configured by IA FW)
Partial Write	Yes (No in BGA congis)

4.4 DDR3L Support

Features	APL
Connection Technology	CBGA
Data Rate (MT/s)	1333, 1600, 1866
DRAM Device Data Width	8 x8, 4 x16
DRAM Device Density	4Gb, 8Gb
Banks / Channel	8
Burst Length	8
Burst Type	Sequential – (Critical 32B First)
Read Latency	6 - 13
Write Latency	6 - 9
Auto Precharge Read/Write	No
Refresh Rate change based on DRAM temperature	Yes, Pcode controlled
Per Bank Refresh	No
Self-Refresh	Yes
Partial Array Self Refresh (PASR)	Yes

Features	APL
Power Down	PPD Slow Exit* or PPD Fast Exit
ZQ Calibration	Yes
Mode Register Access	MR2 Write for extended temperature in SR
CK/CKB Clock Stop	Yes
CK/CKB Clock Tristate	Yes
ODT (Command and Always-on for Write Leveling)	Yes
CA Training	Yes
tCPDED	Minimum 2 DES after PD/SR entry
Write Leveling	Yes
Partial Write	No

4.4.1 Supported DRAM Configurations

If a channel has 2 ranks, the timing parameters must be configured the same for all ranks. The IA FW should configure the memory controller with common timing parameters for both ranks.

4.5 Memory Reference Code

MRC – Memory Reference Code – is the DRAM/Memory-Controller initialization flow performed by IA FW before DRAM is enabled for functional access.

4.5.1 MRC Flow Selection

There are basic types of MRC flows:

1. Cold Boot Flow – Requires full D-unit, DDRIO/PHY, and DRAM init sequence and training.
2. Code Reset Flow - Requires full D-unit, DDRIO/PHY, and DRAM init sequence, but the training could be skipped on the subsequent boot when training data is available.
3. S0i3/S3 Exit Flow – DRAM is expected to contain valid data (and be in Self-Refresh). D-unit and DDRIO/PHY should be configured/restored with values matching the state when S0i3 was entered and the DRAM should be transitioned from the Self-Refresh to the idle state.
4. Warm Reset Flow – DRAM is expected to contain valid data (and be in Self-Refresh). Requires D-unit and PHY reconfiguration but not training or DRAM init sequence.

Cold boot flow should be performed in the following cases:

1. Cold reset – D-unit powergood reset after full power up.
2. Warm reset that occurs after Cold Reset but before MRC is complete.



3. DRAM power is lost/removed during S3.

S3 Exit Flow should be performed in the following cases:

1. S3 Reset – D-unit powergood reset after S3 Exit.

4.5.2 Platform Programming Considerations

Some basic initialization must be performed by IA FW prior to calling the MRC entry points.

The following registers and policy are required to be programmed:

- ECBASE
- SMBUS Base Address (Bus:00 Device:31 Function:01 Register:0x20) this I/O space must be enabled in the SMBUS CMD register (Bus:00 Device:31 Function:01 Register:0x04)
- Graphics UMA and GTT size (Bus: 00 Device:02 Function:00 Register:0x50)
- DRAM_POLICY is a structure that defines the memory configuration that is passed as an input to MRC

4.5.3 MRC Build Flags

There are several build flags that are located in the file .\Mmrc\ProjectData\BXTP\Include\MmrcProjectDefinitionsGenerated.h. These flags can be used to add/remove features from the MRC build. This section describes the flags included in the MRC.

Build Flag	Description
DDR3_SUPPORT	This build flag enables support for DDR3
LPDDR3_SUPPORT	This build flag enables support for LPDDR3
LPDDR4_SUPPORT	This build flag enables support for LPDDR4
MRC_DEBUG_DISABLE	This build flag disable the debug message print.

4.6 BIOS Data Structures

Intel BIOS reference code defines a compatible data structure (BDAT). The IA FW data structure consists of three sections: a compatible header, a versioned data range, and an optional OEM data range. BIOS reference code shall reserve a memory range on the System BSP stack while operating from cache in No-Eviction Mode (NEM). The memory range shall be large enough to store the compatible data structure.

The BIOS reference code shall initialize fields in the data structure, update the CRC value, and store a pointer to the data structure in a scratchpad register in the processor. To determine BIOS execution context, a remote host application can optionally read the current BIOS checkpoint from a separate scratchpad register. The meaning of checkpoints on a given platform shall be consistent within BIOS reference



code, but checkpoints may vary across different System BIOS implementations and different platforms.

After BIOS reference code completes, the System BIOS shall copy the data structure from the NEM stack to an intermediate system memory location and update the pointer in the scratchpad register. The data structure on the NEM stack will be invalidated by tearing down the NEM stack. The System BIOS shall reserve the intermediate memory range using the POST Memory Manager, EFI memory management services, and internal mechanisms so that subsequent steps during BIOS POST do not corrupt the data structure. If the System BIOS needs to update fields within the data structure, it shall be responsible for recalculating the CRC after the updates. The data structure must be relocated to different memory addresses, so pointers to fields within the data structure should be avoided. Instead, offsets can be used relative to the base address of the data structure. External pointers should also be defined as offsets of type UINT32 or UINT64 (relative to base address 0).

When the system BIOS establishes the final system memory map and location of the ACPI memory regions, the data structure shall be copied from the intermediate memory location to its final resting place in AddressRangeReserved, an ACPI Type 2 memory region below 4 GB. The system BIOS must reserve a memory region that is large enough to accommodate the size of the BIOS data structure rounded up to the next 4 KB page size and aligned to a 4 KB address boundary. The system BIOS shall report the physical address range of the Type 2 memory region as required by the ACPI specification. This includes software Interrupt 15h - function AX = E820h, or EFI GetMemoryMap if applicable.

The system BIOS shall initialize a custom ACPI table containing a pointer to the physical base address of the BIOS data structure within the Type 2 memory region. The system BIOS shall then update the scratchpad register with the physical address of the BIOS data structure and free the intermediate memory location. All subsequent accesses to the BIOS data structure should be directed to the final runtime location in the ACPI Type 2 region

4.6.1

Compatible ACPI Table

The pointer to the BIOS data structure shall be defined in a custom ACPI table to provide compatibility across multiple platforms. The Root System Description Table (RSDT) shall reference a custom OEM table identified by the unique signature "BDAT". The BDAT table shall conform to the standard ACPI header and contain a Global Address Structure that defines the 64-bit physical base address of the BIOS data structure. An OS driver may be required to access the custom ACPI table and to load pages containing the BIOS data structure.

```
#pragma pack(1)

#pragma pack(1)

typedef unsigned char      UINT8;
typedef unsigned short     UINT16;
typedef unsigned long      UINT32;
typedef unsigned long long UINT64;

#define EFI_SIGNATURE_16(A, B)      ((A) | (B << 8))
```



```
#define EFI_SIGNATURE_32(A, B, C, D)  (EFI_SIGNATURE_16 (A, B) |  
                                     (EFI_SIGNATURE_16 (C, D) << 16))  
  
//  
// Common ACPI description table header. This structure prefaces most  
ACPI tables.  
//  
typedef struct {  
    UINT32 Signature;  
    UINT32 Length;  
    UINT8 Revision;  
    UINT8 Checksum;  
    UINT8 OemId[6];  
    UINT64 OemTableId;  
    UINT32 OemRevision;  
    UINT32 CreatorId;  
    UINT32 CreatorRevision;  
} EFI_ACPI_DESCRIPTION_HEADER;  
  
//  
// ACPI 3.0 Generic Address Space definition  
//  
typedef struct {  
    UINT8 AddressSpaceId;  
    UINT8 RegisterBitWidth;  
    UINT8 RegisterBitOffset;  
    UINT8 AccessSize;  
    UINT64 Address;  
} EFI_ACPI_3_0_GENERIC_ADDRESS_STRUCTURE;  
  
//  
// BIOS Data ACPI structure  
//  
typedef struct {  
  
    EFI_ACPI_DESCRIPTION_HEADER Header;  
    EFI_ACPI_3_0_GENERIC_ADDRESS_STRUCTURE BdatGas;  
  
} EFI_BDAT_ACPI_DESCRIPTION_TABLE;  
#pragma pack(1)  
  
typedef unsigned char      UINT8;  
typedef unsigned short     UINT16;  
typedef unsigned long      UINT32;  
typedef unsigned long long UINT64;  
  
#define EFI_SIGNATURE_16(A, B)      ((A) | (B << 8))  
#define EFI_SIGNATURE_32(A, B, C, D)  (EFI_SIGNATURE_16 (A, B) |  
                                     (EFI_SIGNATURE_16 (C, D) << 16))  
  
//  
// Common ACPI description table header. This structure prefaces most  
ACPI tables.  
//  
typedef struct {
```



```

    UINT32  Signature;
    UINT32  Length;
    UINT8   Revision;
    UINT8   Checksum;
    UINT8   OemId[6];
    UINT64  OemTableId;
    UINT32  OemRevision;
    UINT32  CreatorId;
    UINT32  CreatorRevision;
} EFI_ACPI_DESCRIPTION_HEADER;

//  

// BIOS Data Parameter Region Generic Address  

// Information  

//  

#define EFI_BDAT_ACPI_POINTER 0x0

//  

// BIOS Data Table  

//  

EFI_BDAT_ACPI_DESCRIPTION_TABLE BiosDataTable = {  

    EFI_SIGNATURE_32 ('B','D','A','T'),           // Signature  

    sizeof (EFI_BDAT_ACPI_DESCRIPTION_TABLE),     // Length  

    0x01,                                         // Revision [01]  

    //  

    // Checksum will be updated during boot  

    //  

    0,                                              // Checksum  

    ' ',                                            // OEM ID  

    ' ',  

    ' ',  

    ' ',  

    ' ',  

    ' ',  

    ' ',  

    0,                                              // OEM Table ID  

    0,                                              // OEM Revision  

    [0x00000000]  

    0,                                              // Creator ID  

    0,                                              // Creator Revision  

    0,                                              // System Memory Address  

    Space_ID  

    0,  

    0,  

    0,  

    // Pointer will be updated during boot  

    EFI_BDAT_ACPI_POINTER,  

};

#pragma pack()

```



4.6.2 Compatible BIOS Structure Header

The following data structure defines the compatible header.

```
#pragma pack(1)

typedef struct {
    UINT8 BiosDataSignature[8];      // "BDATHEAD"
    UINT32 BiosDataStructSize;          // sizeof
} BDAT_STRUCTURE
{
    UINT16 Crc16;                  // 16-bit CRC of BDAT_STRUCTURE
    (calculated with 0 in this field)
    UINT16 Reserved;
    UINT16 PrimaryVersion;           // Primary version
    UINT16 SecondaryVersion;          // Secondary
    version
    UINT32 OemOffset;                // Optional offset
    to OEM-defined structure

    UINT32 Reserved1;
    UINT32 Reserved2;
} BDAT_HEADER_STRUCTURE;

#pragma pack()
```

The signature string shall be initialized to the ASCII sequence "BDATHEAD".

The primary and secondary versions shall uniquely define the format of the data range such that an application can cast the memory range with the associated C structure and decode the data fields. The secondary version number shall be incremented when data fields are appended to the previous version of the data structure. The secondary version number can be recycled when the primary version changes.

The OEM offset provides an optional mechanism for OEMs to customize the BIOS data structure without affecting compatibility of the versioned data range. The version numbers do not apply to the OEM data range, although fields in the versioned data range can be initialized by the OEM system BIOS. The OEM offset is provided mainly as a courtesy for customers that wish to use the BIOS data structure mechanism to transfer information to an OS driver. The format of the OEM data range is outside the scope of this specification.

4.6.3 BIOS Structure Definitions

The following code defines the BDAT_SYSTEM_STRUCTURE.

```
#pragma pack(1)

#define MAX_NODE          4      // Max processors per system
#define MAX_CH            4      // Max channels per socket
#define MAX_DIMM          3      // Max DIMM per channel
#define MAX_RANK_DIMM     4      // Max ranks per DIMM
#define MAX_STROBE         18     // Number of strobe groups
#define MAX_SPD_BYTE       256    // Number of bytes in Serial EEPROM
```



```

typedef struct {
    UINT8    rxDqLeft; // Units = piStep
    UINT8    rxDqRight;
    UINT8    txDqLeft;
    UINT8    txDqRight;
    UINT8    cmdLeft;
    UINT8    cmdRight;
    UINT8    recvenLeft; // Units = recvenStep
    UINT8    recvenRight;
    UINT8    wrLevLeft; // Units = wrLevStep
    UINT8    wrLevRight;
    UINT8    rxVrefLow; // Units = rxVrefStep
    UINT8    rxVrefHigh;
    UINT8    txVrefLow; // Units = txVrefStep
    UINT8    txVrefHigh;
    UINT8    cmdVrefLow; // Units = caVrefStep
    UINT8    cmdVrefHigh;
} BDAT_RANK_MARGIN_STRUCTURE;

typedef struct {
    per rank
        UINT16    recEnDelay[maxStrobe]; // Array of nibble training results
        UINT16    wlDelay[maxStrobe];
        UINT8     rxDqDelay[maxStrobe];
        UINT8     txDqDelay[maxStrobe];
        UINT8     clkDelay;
        UINT8     ctlDelay;
        UINT8     cmdDelay[3];
        UINT8     ioLatency;
        UINT8     roundtrip;
    } BDAT_RANK_TRAINING_STRUCTURE;

typedef struct {
    UINT16    mr0; // MR0 settings
    UINT16    mr1; // MR1 settings
    UINT16    mr2; // MR2 settings
    UINT16    mr3; // MR3 settings
} BDAT_RANK_MRS_STRUCTURE;

typedef struct {
    UINT16    mr0; // MR0 settings
    UINT16    mr1; // MR1 settings
    UINT16    mr2; // MR2 settings
    UINT16    mr3; // MR3 settings
} BDAT_RANK_MRS_STRUCTURE;

typedef struct {
    rank
        UINT8      rankEnabled; // 0 = Rank disabled
        UINT8      rankMarginEnabled; // 0 = Rank margin disabled
        UINT8      dqMarginEnabled; // 0 = Dq margin disabled
        BDAT_RANK_MARGIN_STRUCTURE rankMargin; // Rank margin data
        BDAT_DQ_MARGIN_STRUCTURE dqMargin[maxDq]; // Array of Dq margin data per
        BDAT_RANK_TRAINING_STRUCTURE rankTraining; // Rank training settings
        BDAT_RANK_MRS_STRUCTURE rankMRS; // Rank MRS settings
    } BDAT_RANK_STRUCTURE;

typedef struct {

```



```
    UINT8          valid[MAX_SPD_BYTE/8];           // Each valid bit maps to SPD
byte
    UINT8          spdData[MAX_SPD_BYTE];            // Array of raw SPD data bytes
} BDAT_SPD_STRUCTURE;

typedef struct {
    UINT8          dimmEnabled;                  // 0 = DIMM disabled
    BDAT_RANK_STRUCTURE rankList[maxRankDimm]; // Array of ranks per DIMM
    BDAT_SPD_STRUCTURE spdBytes;                // SPD data per DIMM
} BDAT_DIMM_STRUCTURE;

typedef struct {
    UINT8          chEnabled;                   // 0 = Channel disabled
    UINT8          numDimmSlot; // Number of slots per channel on the
board
    BDAT_DIMM_STRUCTURE dimmList[maxDimm]; // Array of DIMMs per channel
} BDAT_CHANNEL_STRUCTURE;

typedef struct {
    UINT8          imcEnabled;                  // 0 = MC disabled
    UINT16         imcDid;                     // MC device Id
    UINT8          imcRid;                     // MC revision Id
    UINT16         ddrFreq;                    // DDR frequency in units of MHz / 10
                                                // e.g. ddrFreq = 13333 for tCK = 1.5 ns
    UINT16         ddrVoltage;                 // Vdd in units of mV
                                                // e.g. ddrVoltage = 1350 for Vdd = 1.35 V
    UINT8          piStep;                     // Step unit = piStep * tCK / 2048
                                                // e.g. piStep = 16 for step = 11.7 ps (1/128
tCK)
    UINT16         rxVrefStep;                 // Step unit = rxVrefStep * Vdd / 100
                                                // e.g. rxVrefStep = 520 for step = 7.02 mV
    UINT16         txVrefStep;                 // Step unit = txVrefStep * Vdd / 100
    UINT16         caVrefStep;                 // Step unit = caVrefStep * Vdd / 100
    UINT8          recvenStep;                 // Step unit = recvenStep * tCK / 2048
    UINT8          wrLevStep;                  // Step unit = wrLevStep * tCK / 2048
    BDAT_CHANNEL_STRUCTURE channelList[maxCh]; // Array of channels per socket
} BDAT_SOCKET_STRUCTURE;

typedef struct {
    UINT32         refCodeRevision; // Matches JKT scratchpad definition
    UINT8          maxNode;        // Max processors per system, e.g. 4
    UINT8          maxCh;         // Max channels per socket, e.g. 4
    UINT8          maxDimm;        // Max DIMM per channel, e.g. 3
    UINT8          maxRankDimm;   // Max ranks per DIMM, e.g. 4
    UINT8          maxStrobe;      // Number of Dqs used by the rank, e.g. 18
    UINT8          maxDq;          // Number of Dq bits used by the rank, e.g. 72
    UINT32         marginLoopCount; // Units of cache line
    BDAT_SOCKET_STRUCTURE socketList[maxNode]; // Array of sockets per system
} BDAT_SYSTEM_STRUCTURE;

typedef struct bdatStruct {
    BDAT_HEADER_STRUCTURE bdatHeader;
    BDAT_SYSTEM_STRUCTURE bdatSys;
} BDAT_STRUCTURE;

#pragma pack()
```

4.7 System Memory Map

This section describes the system memory map and programming requirements for the memory map registers. Refer APL SoC EDS for register details.

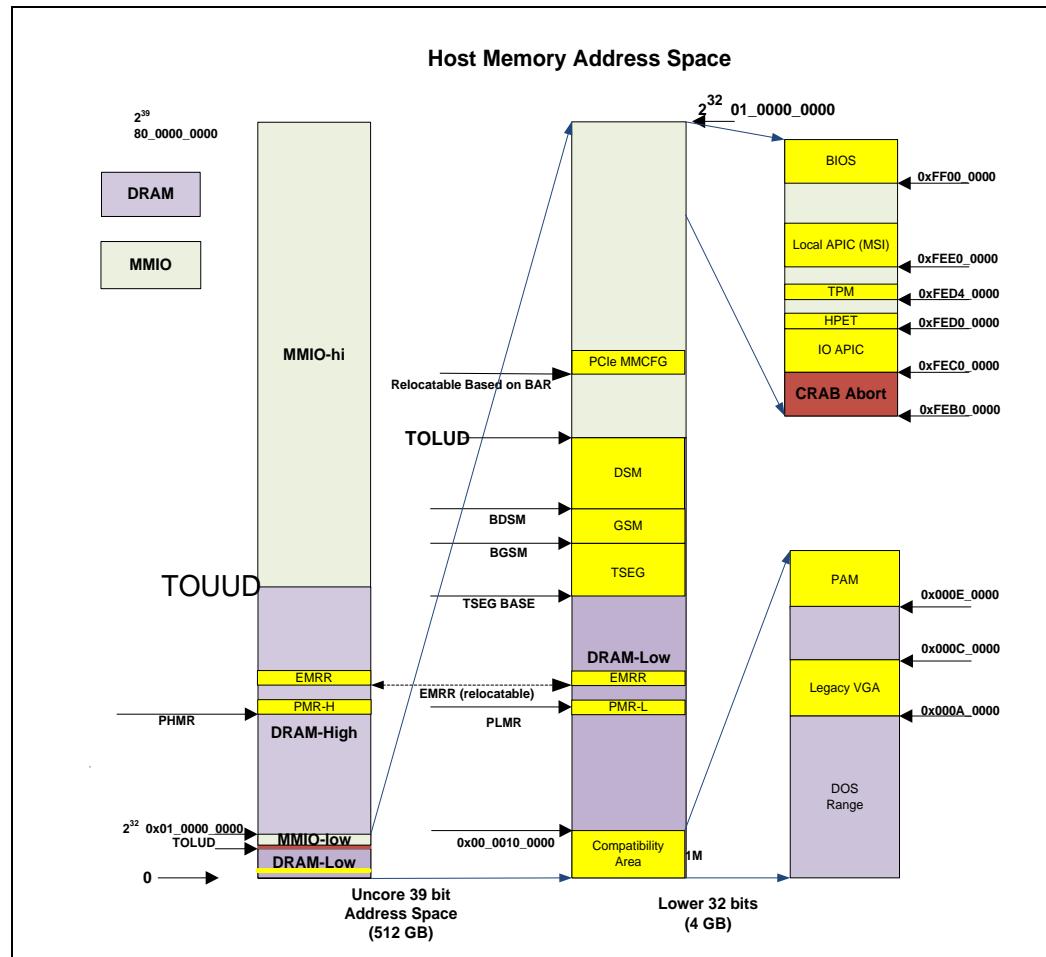


Table 4-1. APL Memory Map Configuration Registers

Register Name	Definition
TOUUD	Top of Upper Usable DRAM. Address above TOUUD target High MMIO. Address from 4GB up to TOUUD target DRAM.
TOLUD	Top of Low Useable DRAM. As a rule addresses below TOLUD target DRAM and addresses between TOLUD and 4 GB target devices in MMIOs space. There are exceptions to the rule.
BDSM	Base of graphics Data Stolen Memory. A region used by the Integrated Graphics Device (IGD). BDSM specifies the base and TOLUD specifies the DSM region.



Register Name	Definition
BGSM	Base of Graphics Stolen Memory. A region used by the IGD. BGSM specifies the base, and BDMS specifies the limit of the GSM region.
TSEGMB	T segment Memory Base. The region used for SMM protected DRAM. TSEGMB specifies the base, and BGSM specifies the limit.
BMISC	Contains fields for enabling various legacy regions including VGA and the IA FW PAM regions.

Table 4-2. APL Memory Map Regions

Address Range	Range	Access	Security
Low DRAM	0 to (TOLUD – 1)	IDI/IOSF all to DRAM	All allowed unless IMR is used to isolate access
Legacy Video Area	A_0000h-B_FFFFh	Configurable to MMIO by reg BMISC :ABSegInDRAM bit	None
Expansion Area	C_0000h-D_FFFFh	No support for ISA extension in SOC. Always map to DRAM	None –unless IMR is configured
PAM Memory Area	E_0000h-F_FFFFh	No support in SOC. Always map to DRAM	None –unless IMR is configured
ISA Hole	15MB-16MB	No support in SOC. Always map to DRAM	None –unless IMR is configured
Protected Memory Range : PMRL	Programmable 1 per VT-D engine, SOC has 2 VT-Ds. SOC has 2 PMR-L which is below 4GB range. 0.0.0.DefVTdBAR.PMRLBA SE to 0.0.0DefVTdBAR.PMRLLIM IT and 0.0.0.GfxVTdBAR.PMRLBA SE to 0.0.0GfxVTdBAR.PMRLLIM IT	DRAM or ABORT based on security	This range must be protected by IMRs to ensure only the IA core has access to this range before VT-d translation is enabled.
DRAM protectable range	NA	Not supported in SOC	NA
TSGE SMM range	Programmable 0.0.0.TSEGMB to (0.0.0.BGSM - 1)	DRAM or ABORT based on security	SMRR



Address Range	Range	Access	Security
Graphics Stolen Memory	Programmable 0.0.0.MCHBAR.BGSM to (0.0.0.MCHBAR.TOLUD - 1)	DRAM or ABORT	SAI = GT, Display
CRAB_ABORT	0xFEBO_0000 – 0xFEBF_FFFF	Abort	None
IOAPIC	0xFEC0_0000 – 0xFECF_FFFF	MMIO	None
HPET	0xFED0_0000 – 0xFED0_33FF	MMIO	None
xTPM	0xFED4_0000 – 0xFED4_0FFF	MMIO –SPI TPM or TXE FTPM	None
Local APIC	0xFEE0_0000 – 0xFEEF_FFFF	MMIO	None
IA FW Range	0xFFFFC_0000 – 0xFFFF_FFFF	Flash/TXE SRAM FFFD_FFFF- 0xFFFF_FFFF always mapped to TXE SRAM in case to verified boot FUSE is set	Post DID IA FW has no access FFFD_FFFF- 0xFFFF_FFFF FFFD_FFFF- rest FF00_0000 is optional and is usable for SPINOR based Flash case.
PCIe Memory Mapped Config Range	BAR: 0.0.0.MCHBAR.PCIEXBAR (size 256MB)	MMIO-PCI config space	None
High DRAM	0x1_0000_0000 – TOUUD	DRAM	None unless IMRS are configured
Protected Memory Range High	PMR-H again 2 , for above 4GB Above 4GB – programmable 0.0.0.DefVTdBAR.PMRHB ASE to 0.0.0DefVTdBAR.PMRHLI MIT and 0.0.0.GfxVTdBAR.PMRHB ASE to 0.0.0GfxVTdBAR.PMRHLI MIT	DRAM	None
High MMIO Address Range	TOUUD – 0x7F_FFFF_FFFF	MMIO	None

NOTE: TOUUD_HI should be always set to 0x1 no matter what size memory installed. If TOUUD_HI is set to 0x0, TXE FW will initiate a cold reset to stop system booting.

§





5 Internal Graphics Configuration

This chapter describes what IA FW must do to initialize and provide interfaces to the internal graphics functionality in Apollo Lake platform. It highlights the various sections of memory that are needed to be “stolen” (pre-allocated) and the modifications are necessary in the IA FW.

5.1 Internal Graphics Enabling

To enable internal graphics, the IA FW must perform the following steps:

1. Verify internal Graphics is supported on the processor by verifying D0.F0.R0E4h [11] = ‘0b’. If the bit is set to ‘1b’, internal Graphics is not supported.
2. Perform the following steps only if Internal Graphics is supported.
3. Program D0.F0.R054h [4] = ‘1b’ to enable Internal Graphics Device.
4. Enable Internal Graphics Device VGA Decode by setting D0.F0.R050h [1] = ‘0b’.

5.2 Graphics Memory Requirements

	Register	Value Range
DSM	B0/D0/F0:R 50h [15:8] B0/D0/F0:R B0h	0 - 2048 MB
GSM	B0/D0/F0:R 50h [7:6] B0/D0/F0:R B4h	0/2MB/4MB/8MB
GTTMMADR	B0/D2/F0:R 10h	8 MB/10 MB/12 MB/16 MB
GMADR	B0/D2/F0:R 62h	128 MB/256 MB/512 MB/1 GB/2 GB/4 GB

For Apollo Lake, DSM size can vary between 0 to 2048 MB and IA FW needs to program DSM size by setting D0.F0.R050h [15:8] - Graphics Mode Select (GMS) to 01h to support default value of 32 MB pre-allocated Graphics memory.

- Program graphics data stolen memory (DSM) base in D0.F0.R0B0h [31:20]. In general the DSM base address is derived by IA FW from the equation
$$\text{DSM Base address} = \text{TOLUD} - \text{DSM size}$$
- Program GTT stolen memory size (GSM) by setting D0.F0.R050h [7:6] - GGMS (GTT Graphic Memroy Size) to 2 MB, 4MB or 8 MB. 1 MB is no more a valid configuration. Recommended default configuration for GTT size on Apollo Lake is 8MB (0011b).
- The GTT memory base address is configured by IA FW in D0.F0.R0B4h. In general the GTT base address is derived by IA FW from the equation
$$\text{GTT Base address} = \text{DSM base} - \text{GTT size}$$



- IA FW must configure the GMADR aperture size and base address prior to enumeration of Device 2 registers. Configuring register D2.F0.R062h [4:0] sets the aperture size and this determines which bits of D2.F0.R018h [38:32] actually correspond to GMADR base address. Recommended default aperture size for Apollo Lake is 256 MB (00001b).
- IA FW must configure the GTTMMADR base address is D2.F0.R010h. On Apollo Lake, the GTTMMADR region size can be configured to 8 MB/10 MB/12 MB/16MB. However the address must be aligned to a 16 MB boundary and a default value of 16 MB is recommended for Apollo Lake. The GTTMMADR base address is derived by IA FW as follows.

$$\text{GTTMADR} = (\text{2 MB MMIO} + \text{6 MB Reserved} + \text{GTT size}) \text{ aligned to 16 MB.}$$

Note: The programming steps above should be done prior to running any PCI VGA ROM code cycles and Video BIOS Initialization code and also before memory initialization. The size of "pre-allocated" memory is programmed by IA FW depending on OEM platform requirements.

5.3 Dynamic Video Memory Technology

Dynamic Video Memory Technology (DVMT) is a technology that ensures the most efficient use of available memory for balanced 2D/3D Graphics performance and system performance. Graphics DVMT dynamically responds to system requirements and application demands, by allocating the proper amount of display, texturing, and buffer memory after the operating system has booted.

Apollo Lake Graphics only supports DVMT 5.0 which has the following advantages over the previous DVMT versions:

- Pre-allocated memory provides the system-guaranteed video memory under all conditions.
- Allow scalability of Graphics memory as system memory size increases in the system.

Refer Intel® HD Graphics – Dynamic Video Memory Technology (DVMT) 5.0 White Paper (IBL# 444997) for more details.

5.4 Internal Graphics Disabling

If internal Graphics is to be disabled, the IA FW must perform the following steps:

1. Set the GMS field of the GMCH Graphics Control Register D0.F0.R050h [7:6] = '00b'. This prevents UMA memory from being pre-allocated to the internal Graphics device.
2. Set the GGMS field of the GMCH Graphics Control Register D0.F0.R050h [15:8] = '00000000b' and set the internal Graphics device VGA Disable (IVD) field of the GMCH Graphics Control Register D0.F0.R050h [1] = '1b'. This will prevent the internal Graphics device from claiming VGA cycles.
3. Program D0.F0.R054h [4] = 0b to disable internal Graphics.

5.5

VBIOS/ GOP Driver Post Time Reduction

VBIOS/ GOP Driver spend T3 time for eDP panel power up as required by specification. For system uses eDP panel as primary display, it is possible to avoid this delay by IA FW ensuring GTTMMADR is properly configured in early boot phase (PEI) to turn on the eDP panel power.

5.6

GT Power Management

For an Intel GFX enabled platform, IA FW is expected to perform the below sequences in order to lock down critical settings. These are functions that impact GTs reporting of actual power impact, or the ability of the PCU to limit GT's power consumption/thermal impact.

IA FW must execute the following steps after Memory Initialization is complete.

- Set D2.F0.R004h [2:0] = '111b' to enable Memory Access Enable, Bus Master Enable and I/O Access Enable.
- Configure the GTTMMADR base address in D2.F0.R010h.

5.6.1

RC6 Enabling

IA FW must follow the below steps to enable RC6 feature.

Step	Type	Address	Data	Comment
1	PCI 0.2.0	<MMADR>	<MMADR Base>	Should already be set at PCI enumeration
1a	MMIO	0xD40	0x80000001	RC6 Context Location [31]=1 (Lock) [0]=1 (DRAM)
1b	MMIO	0xD48	<Calculate>	Context Base [31:12] = Base[31:12] [11:4] = Base[39:32] (0x00 since stolen mem below 4GB) [2] = 1 [1] = 1 (DRAM only) [0] = 1 (Lock address) Must set to a physical address within GT stolen Memory WOPCM, at least 24KB from the top. This range must be coordinated with other uses (like GuC and PAVP); it is expected that the upper 24KB of stolen memory is the proper location
1c	MMIO	0xD08	0x00000000	Clock gating disables. Enable all GTi-Uncore clock gating (they default to '1' = disabled)



Step	Type	Address	Data	Comment
2a	MMIO	0xA188	0x00010001	Set Force Wake =1
2b	MMIO	0x130044	Read	Poll to verify Force Wake Acknowledge bit Poll until [0] = 1
3a	MMIO	0xA250	0x000001FF	Push bus Metric Counter Enable [8:0]=111111111: Enable all counters
3b	MMIO	0xA25C	0x00000010	Push bus Shift [5:4]=01: Shift C0 residency 1 bit All others 00 = no shift
3c	MMIO	0xA248	0x80000004	Push bus Metric Control [31]=1: Lock (may be 0 for debug BIOS) [6:3]=0000: Include all engines in C0 residency [2]=0: C0 residency counts clocks (vs time) [1]=0: Video busy counts clocks (vs time) [0]=0: Video Busy select (instead of C1 residency)
4	MMIO	0xA000	0x00070020	GFXPAUSE [18]=1: Lock (may be 0 for debug BIOS) [17:16]=11: Enable EU Pause and Sampler Stall [15:0]=32d: Pause count 32 csclks.
5a	MMIO	0xA180	0xC5200000	GPM Control [31]=1: Lock to prevent changes (optional during debug) [30]=1: Treat all CPDs the same (no IA shortcut) [29]=0: Do not block pipes on CPD [28]=0: Do not do Funny-IO Core Status writes during CPD [27]=0: Allow CoreActive to flow from SQ status [26]=1: Discard non-shadowed cycles when in C6 [25]=0: No Go=0 during CPD [24:22] =100: After FLR, go to C6



Step	Type	Address	Data	Comment
				[21]=1: All agents must be idle for C6 [20]=0: Primary NOA
5b	MMIO	0x9424	0x0000007BD (A stepping) 0x0000007FD (B0+ steppings)	Clocking/Reset controls [10]=1: Enable Media Sampler DOP gating [9]=1: Enable SFC2 DOP Gating [8]=1: Enable SFC1 DOP Gating [7]=1: Enable VE DOP gating [6]=1: Enable Media DOP gating (0 for A stepping) [5]=1: Enable Media1 DOP gating [4]=1: Enable GuC DOP gating [3]=1: Enable WiDi DOP gating [2]=1: Enable FF DOP gating [1]=0: Enable L1 clock gating during reset (still needed?) [0]=1: Enable Render DOP gating
5c	MMIO	0x9400	0x00000000	Enable unit level clock gates (these may change in later steppings as bugs are fixed) [25]=0: Enable GTI unit clock gating [23]=0: Enable GPM unit clock gating [22]=0: Enable GAM/GAMW unit clock gating [21]=0: Enable GAC unit clock gating [20]=0: Enable GAB unit clock gating
5d	MMIO	0x9404	0x40401000	Enable unit level clock gates (these may change in later steppings as bugs are fixed)
5e	MMIO	0x9408	0x00000000	Enable unit level clock gates (these may change in later steppings as bugs are fixed)
5f	MMIO	0x940c	0x02000001	Enable unit level clock gates (these may change in later steppings as bugs are fixed)
5g	MMIO	0x9430	0x10004000 (A stepping)	[28] = 1: HDCREQ Clock Gating Disable for all steppings



Step	Type	Address	Data	Comment
			0x10000000 (B0+steppings)	[14] = 1: SDE units Clock Gating Disable for A0 only
6	MMIO	0x2054 0x12054 0x22054 0x1A054 0x1C054 0xC3E4	0x0000000A	Set all *CS/GuC Idle Max Count (hysteresis) [9:0]=10d: Set to 5us (roughly)
7	MMIO	0x2050 0x12050 0x22050 0x1a050 0x1c050	0x00010000	Enable Idle Messages from all *CS
8a	MMIO	0xA008	0x06000000	Set Normal frequency request: [31:23] : Request Unslice ratio [22:14] : Request Slice ratio [3:0]=0000 : Policy: Slice uses Unslice frequency
8b	MMIO	0xA024	0x00000592	RP Control [11]=0: Video Turbo when Media engine busy [10:9]=10: Use Normal frequency req [8]=1: Media engine counts toward gfx busyness [7]=1: RP Counter enable [5:3]=010: Up=Busy Max Average (EI) [2:0]=010: Down=Busy Min Average (EI)
9	MMIO	0xA094	0x00040000	Software RC state – RC6 This will cause GT to enter RC6 when idle
10a	MMIO	0xA188	0x00010000	Set Force Wake =0 (clear)
10b	MMIO	0x130044	Read	Force Wake Acknowledge bit in System Agent Poll until [0] = 0

5.7 Display CD Frequency Selection

Display CD frequency is default operated at 675MHz. Refer [Table 5-1](#) below to program GTTMMADR offset 46000h [10:0] for CD Clock Frequency.

Table 5-1. CD Frequency Configuration

Display Frequency	GTTMMADR 46000h [10:0]
144 MHz	0x11E
288 MHz	0x23E
384 MHz	0x2FE
576 MHz	0x47E
624 MHz	0x4DE

5.8 Multi-Monitor Support

If internal Graphics is the primary VGA display adapter and there is another display adapter in the system, nothing else needs to be done. In the case that internal Graphics is the secondary display adapter (i.e., the PCI Express* video card is the primary), IA FW must perform the following steps:

1. Configure and enable primary VGA display adapter normally.
2. Initiate VBIOS POST on the primary VGA display adapter normally.
3. Set GGC D0.F0.R050h [1] = '1b'. This changes the internal device class-code to 0380h (non-VGA).
4. Configure and enable the iGFX device (B0.D2).
5. Copy VBT data of internal (iGFX) VBIOS to OpRegion Mailbox4.

5.9 Video BIOS and Video Driver Interface to IA FW

In order to achieve compatibility with the Video BIOS and Graphics Driver, the IA FW should implement ASL code methods to comply with the OpRegion/GMCH SCI Specification. This specification defines a mechanism by which IMC Video Drivers and Video BIOS can work as clients of the IA FW, thus allowing the IA FW to notify the software of events, and using the IA FW to perform platform-specific functions which could not otherwise be performed directly by the Video Driver and/or Video BIOS without detailed knowledge of the platform-specifics. Aslo, this specification avoids replicating the platform-specific interfaces by providing for a single common method which both Video BIOS and Drivers can use.

5.10 UMA Memory Space Cache-Ability

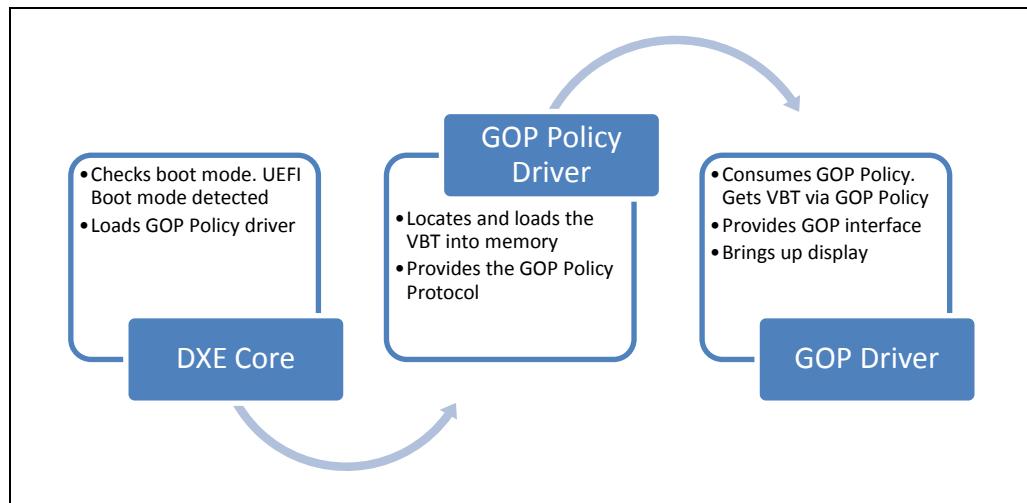
Most of the memory testing software available executes write/read sequences to memory to check how much memory is present in the platform. This causes issues when the “pre-allocated” memory space is left as cached. So IA FW needs to ensure that the memory space that is “pre-allocated” from physical memory needs to be left Uncached by using the Processor MTRR’s appropriately.

5.11 GOP Driver Implementation

Graphics Output Protocol Driver (GOP Driver) is a native UEFI driver that enables GOP, as defined in UEFI 2.3 Specification to support graphics console output in the pre-OS phase. GOP Driver is accessed through UEFI protocols, whereas legacy VBIOS is accessed through interrupts and VGA/VBE interface. GOP Driver is available only during boot services, whereas legacy VBIOS is available during both boot and OS runtime services. UEFI GOP Driver will be released into two parts, one is GOP Driver binary file another is VBT.bin file. IA FW needs to include these two files into IA FW binary file and create a policy driver for GOP Driver. The “VBT.bin” file will be used by the BMP tool for OEM customization and IA FW should return the pointer to this file through the policy protocol.

If UEFI boot mode is detected during system boot, then DXE Core will load the GOP Policy driver. GOP Policy driver locates and loads VBT into local system memory. GOP Driver consumes GOP Policy, gets VBT through GOP Policy, provides GOP interface and brings up the display.

Figure 5-1. UEFI Boot Mode



5.12 Protected Audio Video Playback (PAVP)

PAVP is required to enable the playback of the new high-definition media as described by the Blu-Ray Disc standard. The standard uses a common content management and protection system specified by the Advanced Access Content System (AACS). AACS

has increased robustness requirements for the playback environment than the CSS license used by standard definition (regular) DVD disks. IA FW is required to support PAVP for Blu-ray playback. PAVP enabled software players will not be able to play Blu-ray disc content if IA FW PAVP support is not implemented.

5.12.1 PAVP Configuration Steps

This section explains the procedure to enable PAVP support in IA FW.

1. To allocate Protected Content memory base, configure the size in register B0.D0.F0.R058h [31:20]. Because Apollo Lake does not support 256 KB PCM size and 2 MB, 4 MB and 8 MB are new on Apollo Lake. It is recommended to configure PCM size as 1 MB. The effective Protected Content memory base is X MB below the TOLUDBASE where x means the allocated PCM Base size.
2. Set register B0.D0.F0.R058h [2:0] = '111b' to enable the PCM, PAVP and lock down configurations.

5.13 Access to MMIO Registers in SMM Mode

The IA FW will support a special Video BIOS alternate entry point when invoking the Video BIOS from within the SMM handler. This feature is necessary due to the heavy use of MMIO only controller registers. The MMIO registers are mapped to the display memory A0000:BFFFh aperture so the real mode ROM Video BIOS can access them. However, this presents a problem in SMM mode when the SMBASE for SMRAM is set to A0000h. The Video BIOS will not have access to the register space for save/restore, switch display device, and other functions.

To solve this problem, the Video BIOS will require the IA FW to use an alternate interrupt 10h entry point to the Video BIOS dispatcher. This alternate entry point (WORD offset from start of BIOS) is located at 12h which bypasses the dispatcher's STI instruction and sets an internal SMM flag (GR18[0]). Besides calling this special interrupt 10h entry point the IA FW must leave the System State as 32-bit real mode. With these two requirements the Video BIOS can access the MMIO registers in the extended memory space.

5.13.1 Memory Map Register Programmed and Locked

As part of the memory map setup, IA FW should program and lock (if lock-able) the following registers.

Table 5-2. IA FW Programmed Memory Related Registers

Address BDF : Offset	Description
Device 2 Cfg Registers	
0:2:0:10-17h	GTTMMADR – Gfx Memory Mapped Address Register. This is the base address for memory mapped registers and GTT table.
0:2:0:18-1Fh	GMADR – Gfx Memory Address Range (ie. Gfx Aperture)



0:2:0:20-23h	IOBAR - I/O Base Address. This is used only by IA FW. This register is the base address for the MMIO_INDEX and MMIO_DATA registers
0:2:0:2C-2Dh	SSID – Subsystem Vendor Identification. This register is used to uniquely identify the subsystem where the PCI device resides.
0:2:0:2E-2Fh	SID – Subsystem ID. This register is used to uniquely identify the subsystem where the PCI device resides.
0:2:0:3C	Interrupt Line Register (Intrline)
0:2:0:62h	MSAC – Multi-Size Aperture Control. This register determines the size of the graphics memory aperture. Only the IA FW will write this register based on pre-boot address allocation efforts. Graphics may read this register to determine the correct aperture size. IA FW needs to save this value on boot so that it can reset it correctly during S3 resume.
Device 0 Cfg Registers Shadowed to Device 2	
0:0:0:50-51h	MGGC. Graphics Control. Bit0 of each register is the lock bit.
0:0:0:58-5Bh	PAVPC – Protected Audio Video Control. IA FW will program this register for VLV (not the GFX driver) Bit0 of each register is the lock bit.
0:0:0:5C-5Fh	DPR
0:0:0:A8-AFh	TOUUD – Top of Upper Usable DRAM
0:0:0:B0-B3h	BDSM – Base of Data Stolen Memory. Contains the base address of Graphics Data Stolen DRAM memory. The GTT range and the Data range must be physically contiguous. This is a IA FW responsibility and is noted here because it allows for a single IMR for both. Bit0 of each register is the lock bit.
0:0:0:B4-B7h	BGSM – Base of GTT Stolen Memory. Base of GTT table in Gfx Stolen Memory. The GTT range and the Data range must be physically contiguous. This is a IA FW responsibility and is noted here because it allows for a single IMR for both. Bit0 of each register is the lock bit.
0:0:0:B8-BBh	TSEGMB
0:0:0:BC-BFh	TOLUD – Top of Upper DRAM
MCHBAR Registers	
0:0:0:BAR=0:0x6C88-0x6C8Fh	Gfx VtdBAR



CR Registers	
	Ucode would write relevant Mcheck related registers and trigger the Mcheck flow – for secure boot purposes.

§



6 PMC

PMC is a PCI device with B/D/F = 0/13/1 and this device represents address space for IPC1/GCR/ACPI block.

The PMC function contains three BARs:

1. BAR0 (8KB) – Lower 4KB for IPC1, upper 4KB for GCR (GCR BAR aka PBASE).
2. BAR1 (4KB) – PCI configuration space aliased as MMIO in ACPI mode.
3. BAR2 (I/O 128B) – ACPI configuration.

After programming these BARs, IA FW should configure the PMC function as ACPI function.

6.1 Configure PCI Function as ACPI Device

Below is the flow for IA FW to program and configure,

1. Program the PCI config space (MSE, BAR, Interrupt enable, PMCSR.PowerState = D0, etc)
2. Program the PSF CfgDis bit in the AGNT_T0_SHDW_CFG_DIS register associated with the hybrid ACPI configuration object in the fabric
3. Program the IOSF2OCP Bridge private register PCI_CFG_DIS bit in the PCICFGCTRL register.
4. Program the BAR1_Disable to 0 (enable BAR1) in the PCICFGCTRL register.

6.2 PMC I/O BAR

Programming PMC BAR2 requires a specific methodology. The 128B PMC I/O BAR must be programmed by IA FW using a single configuration write for a 16-bit address on primary to offset 0x20 at B0:D13:F1.

IA FW should assume the BAR2 size is 128B and not depend on a configuration read. The I2O Bridge doesn't support a 3rd BAR and will not forward a configuration write for PMC BAR2 to the OCP fabric but will return a successful completion to IA FW. PSF3 will successfully update its shadowed BAR with the configuration write, which is the desired outcome.

Once I/O BAR2 is programmed in PSF3, PSF3 will route I/O requests targeting the BAR to I2O Bridge, which will subtractive forward to the OCP fabric and PMC, successfully accessing its ACPI configuration.

6.3 ACPI Space

ACPI space is accessed via BAR2 which is an IOBAR of 128B and BIOS fix an IO base address like 0X400 (A configuration read always returns all zeros). Registers in this



space are either directly used by BIOS or its reported OS via standard ACPI FADT table like the PM1 reg/GPE reg etc.

6.4 GCR Space

The GCR (Global Control Register) configuration space begins at offset 0x1000 of BAR0. The majority of these registers are meant for IA FW usage only., please refer to "APL SoC External Design Specification (EDS)" vol2 chapter 15 for GCR register details.

6.5 IPC1

IA FW used IPC1 memory mapped registers to send commands to the ARC. The interface defined below, and the working mode is the following:

4. The IA FW writes the IPC_CMD command register. The category of IPC1 is identified by the IPC_CMD.CMD_ID, and any sub command is identified by the IPC_CMD.SUB_CMD_ID.
5. The IA write to IPC_CMD command register triggers an interrupt to the ARC.
6. The ARC handles the interrupt and services it, writing optional data to the IPC1 registers.
7. The ARC updates the IPC_STS response register with the status of the transaction.
8. If IPC_CMD.MSI was requested, IPC_STS.IRQ will be asserted which causes an interrupt to be sent.
9. IA FW was either polling on IPC_STS.BUSY clearing, or got an interrupt. It reads IPC_STS, and IPC_RBUF if applicable.
10. If an interrupt was sent, IA FW must write '1 to IPC_STS.IRQ to clear the interrupt.

Message Specific Details:

- SUB_CMD "South IP UnGate" will be sent by HOST in preparation for VISA programming. In response PMC FW will send IP_WAKE signals to GMM, USB_XDCI, USB_XHCI, MEX, CSE, P2SB, SCC, LPSS, CUNIT and also do the SPI workaround. Once all actions have been completed (or error detected), PMC FW will update IPC_STS response register.
- SUB_CMD "South IP Allow Gating" will be sent by HOST after VISA has been programmed. PMC FW will unwind the "South IP Ungate" actions. Once unwind has been completed (or error detected), PMC FW will update IPC_STS response register.
- FW puts in a check which makes sure that "South IP UnGate" is called before "South IP Allow Gating" otherwise it errors out. It also errors out if "South IP UnGate" is called consecutively without completing the cycle of ungating IPs and then allowing them to gate.



6.5.1 IPC1 Messages

The IPC1 configuration space begins at offset 0x0000 of BAR0, please refer to "APL SoC External Design Specification (EDS)" vol2 chapter 15 for IPC register details.

Table 6-1. IPC1 Message Summary

CMD_ID	Name	Description
FFh	PMIC Register Access	Allows read and write access to PMIC registers.
F0h	USB 3.3V ON, OFF	Cause PMC to turn on, off USB 3.3V rail
EFh	PMIC Blacklist Select	One-time message to tell PMC to switch from initial PMIC blacklists to Runtime PMIC blacklists
EEh	PHY Config	Tells PMC about PHY configuration – PHY configured for normal operation or disabled and put in low power configuration
EDh	NPK Enable, Disable	Causes North Peak enable, disable during S0
ECh	PM Debug	Allows read, write, and report of LTR values
EBh	PMC Telemetry	Allows read, write to PMC telemetry registers
EAh	PMC FW MSG Control	To read and write the PMC FW_MSG_CTL register
E9h	South IP Gating	To gate, ungate South IPs for VISA programming
E8h	Emi/Rfi Support	To configure LJPLL registers
E7h	Get PMC FW Version	The PMC Firmware version is returned
E5h	Capsule Update Reset	For capsule update reset to clear eMMC write protect

6.5.2 IA FW Requirement

6.5.2.1 PMC FW Version

The message is to provide access to IA to PMC firmware version. BIOS may wish to know and display which version of the PMC firmware which is running.

PMC firmware returns the version to the requestor. Note that the full version number is comprised of the PMC firmware minor version, major version, and engineering version and returned via the read buffer, IPC_RBUF.



IPC_CMD Register

Bits	Description
31:24	Reserved.
23:16	Size. Size of the message data in bytes. Matches number of bytes in IPC_WBUF. 0x0
15:12	Sub Command ID (SUB_CMD_ID) 0x0 – Read PMC firmware version
11:9	Reserved
8	MSI. Generate MSI or not for the message. The state of this bit is left to the host driver—setting this bit allows the driver to be notified using interrupt on completion.
7:0	Command ID (CMD_ID) 0xE7

IPC_STS Register

Bits	Description													
31:24	Reserved													
23:16	Error Code (ERROR CODE). Indicates the Message-specific error code if ERROR is high. <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">0 = none</td><td style="padding: 2px;">No error</td></tr> <tr> <td style="padding: 2px;">1 = command not supported</td><td style="padding: 2px;">IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.</td></tr> <tr> <td style="padding: 2px;">2 = command not serviced</td><td style="padding: 2px;">IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc.).</td></tr> <tr> <td style="padding: 2px;">3 = unable to service now</td><td style="padding: 2px;">IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.</td></tr> <tr> <td style="padding: 2px;">4 = command invalid</td><td style="padding: 2px;">IPC_CMD.CMD_ID is invalid</td></tr> <tr> <td style="padding: 2px;">5 = command failed</td><td style="padding: 2px;">Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.</td></tr> </table>		0 = none	No error	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc.).	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.	4 = command invalid	IPC_CMD.CMD_ID is invalid	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.
0 = none	No error													
1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.													
2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc.).													
3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.													
4 = command invalid	IPC_CMD.CMD_ID is invalid													
5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.													
15:8	Initiator ID (INIT_ID). Hardware updates this value with the ID of the initiator.													

7:4	Command ID (CMD_ID) . Hardware copies value from SUB_CMD_ID field from IPC_CMD register.
3	Reserved
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1' to this bit to clear it and de-assert the interrupt.
1	Error (ERROR) 0 = No error 1 = Error
0	Ready/Busy (BUSY) 0 = IPC is ready to accept next IPC transaction 1 = IPC is busy

IPC_RBUF Register – For all IPC_CMD.SUB_CMD_ID value

Byte	Description
15:8	NA for this IPC_CMD.CMD_ID
7:4	PMC firmware engineering version
3:2	Reserved
1	PMC firmware major version
0	PMC firmware minor version

6.5.2.2 PMC Telemetry Register Access

This message is used to read and write PMC telemetry registers.

IPC_CMD Register									
Bits	Description								
31:24	Reserved.								
23:16	Size . Size of the message data in bytes. Matches number of bytes in IPC_WBUF required for the associated IPC_CMD.SUB_CMD_ID. <table border="1" style="margin-left: 20px;"> <tr> <th>SUB_CMD_ID</th> <th>Size</th> </tr> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>3</td> </tr> <tr> <td>2</td> <td>0</td> </tr> </table>	SUB_CMD_ID	Size	0	1	1	3	2	0
SUB_CMD_ID	Size								
0	1								
1	3								
2	0								



IPC_CMD Register		
Bits	Description	
31:24	3	0
	4	4
	5	0
	6	4
	7	0
	8	4
15:12	Sub Command ID (SUB_CMD_ID)	
	0: TELEM_EVENT_READ	Reads a TELEM_EVENT register.
	1: TELEM_EVENT_WRITE	Writes a TELEM_EVENT register.
	2: TELEM_INFO_READ	Reads the TELEM_INFO register.
	3: TELEM_TRACE_READ	Reads the MBB_TELEM_TRACE_MASK register.
	4: TELEM_TRACE_WRITE	Writes the MBB_TELEM_TRACE_MASK register.
	5: TELEM_TRACE_CTL_READ	Reads the MBB_TELEM_CONTROL register.
	6: TELEM_TRACE_CTL_WRITE	Writes the MBB_TELEM_CONTROL register.
	7: TELEM_EVENT_CTL_READ	Reads the TELEM_EVENT_CTL register.
	8: TELEM_EVENT_CTL_WRITE	Writes the TELEM_EVENT_CTL register.
11:9	Reserved	
8	MSI . Generate MSI or not for the message. The state of this bit is left to the host driver—setting this bit allows the driver to be notified using interrupt on completion.	
7:0	Command ID (CMD_ID) 0Xeb	

IPC_STS Register		
Bits	Description	
31:24	Reserved	

IPC_STS Register																					
Bits	Description																				
23:16	<p>Error Code (ERROR_CODE). Indicates the Message-specific error code if ERROR is high.</p> <table border="1"> <tr> <td>0 = none</td><td>No error</td></tr> <tr> <td>1 = command not supported</td><td>IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.</td></tr> <tr> <td>2 = command not serviced</td><td>IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).</td></tr> <tr> <td>3 = unable to service now</td><td>IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.</td></tr> <tr> <td>4 = command invalid</td><td>IPC_CMD.CMD_ID is invalid</td></tr> <tr> <td>5 = command failed</td><td>Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.</td></tr> <tr> <td>6 = Invalid TELEM_EVENT register number</td><td>The register number specified in a TELEM_EVENT_READ or TELEM_EVENT_WRITE was >= TELEM_INFO.NENABLES</td></tr> <tr> <td>7 = Invalid TELEM_EVENT_CTL programming</td><td>Erroneous programming of TELEM_EVENT_CTL.</td></tr> <tr> <td>8 = TELEM_EVENT ID code event resource conflict</td><td>If there is a resource conflict specifically if there is a HW Counter overflow. For eg: when the programmed widget number is more than or equal to the number of widgets available.</td></tr> <tr> <td>9 = Invalid register access</td><td>Attempted to set a field in a control register while the service described by the control register was enabled</td></tr> </table>	0 = none	No error	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.	4 = command invalid	IPC_CMD.CMD_ID is invalid	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.	6 = Invalid TELEM_EVENT register number	The register number specified in a TELEM_EVENT_READ or TELEM_EVENT_WRITE was >= TELEM_INFO.NENABLES	7 = Invalid TELEM_EVENT_CTL programming	Erroneous programming of TELEM_EVENT_CTL.	8 = TELEM_EVENT ID code event resource conflict	If there is a resource conflict specifically if there is a HW Counter overflow. For eg: when the programmed widget number is more than or equal to the number of widgets available.	9 = Invalid register access	Attempted to set a field in a control register while the service described by the control register was enabled
0 = none	No error																				
1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.																				
2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).																				
3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.																				
4 = command invalid	IPC_CMD.CMD_ID is invalid																				
5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.																				
6 = Invalid TELEM_EVENT register number	The register number specified in a TELEM_EVENT_READ or TELEM_EVENT_WRITE was >= TELEM_INFO.NENABLES																				
7 = Invalid TELEM_EVENT_CTL programming	Erroneous programming of TELEM_EVENT_CTL.																				
8 = TELEM_EVENT ID code event resource conflict	If there is a resource conflict specifically if there is a HW Counter overflow. For eg: when the programmed widget number is more than or equal to the number of widgets available.																				
9 = Invalid register access	Attempted to set a field in a control register while the service described by the control register was enabled																				
15:8	Initiator ID (INIT_ID) . Hardware updates this value with the ID of the initiator.																				
7:4	Command ID (CMD_ID) . Hardware copies value from SUB_CMD_ID field from IPC_CMD register.																				
3	Reserved																				
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1 to this bit to clear it and de-assert the interrupt.																				
1	<p>Error (ERROR)</p> 0 = No error 1 = Error																				
0	<p>Ready/Busy (BUSY)</p> 0 = IPC is ready to accept next IPC transaction 1 = IPC is busy																				



**IPC_WBUF Register – when IPC_CMD.SUB_CMD_ID is 0X1
(TELEM_EVENT Write)**

Byte	Description
15:3	NA for this IPC_CMD.SUB_CMD_ID
2:1	Value to be written
0	Number of the TELEM_EVENT register to write

**IPC_WBUF Register – when IPC_CMD.SUB_CMD_ID is 0X0
(TELEM_EVENT Read)**

Byte	Description
15:1	NA for this IPC_CMD.SUB_CMD_ID
0	Number of the TELEM_EVENT register to read

**IPC_WBUF Register – when IPC_CMD.SUB_CMD_ID is 0X4,
0x6, 0x8
(TELEM_TRACE Write, TELEM_TRACE_CTL Write,
TELEM_EVENT_CTL Write)**

Byte	Description
15:4	NA for this IPC_CMD.SUB_CMD_ID
3:0	Value to be written to the register

**IPC_RBUF Register - when IPC_CMD.SUB_CMD_ID is 0X0
(TELEM_EVENT Read)**

Description
NA for this IPC_CMD.SUB_CMD_ID
Data read from the TELEM_EVENT register

IPC_RBUF Register - when IPC_CMD.SUB_CMD_ID is 0x2, 0x3, 0x5, 0x7 (TELEM_INFO Read, TELEM_TRACE Read, TELEM_TRACE_CTL Read, TELEM_EVENT_CTL Write)	
Byte	Description
15:4	NA for this IPC_CMD.SUB_CMD_ID
3:0	Data read from the register

6.5.2.3 South IP Ungate

- SUB_CMD “South IP UnGate” will be sent by HOST in preparation for VISA programming. In response PMC FW will send IP_WAKE signals to USB_XDCI, USB_XHCI, MEX, CSE, P2SB, SCC, LPSS, CUNIT and also do the SPI workaround. Once all actions have been completed (or error detected), PMC FW will update IPC_STS response register.
- SUB_CMD “South IP Allow Gating” will be sent by HOST after VISA has been programmed. PMC FW will unwind the “South IP Ungate” actions. Once unwind has been completed (or error detected), PMC FW will update IPC_STS response register.
- FW puts in a check which makes sure that “South IP UnGate” is called before “South IP Allow Gating” otherwise it errors out. It also errors out if “South IP UnGate” is called consecutively without completing the cycle of ungating IPs and then allowing them to gate.

IPC_CMD Register	
Bits	Description
31:24	Reserved.
23:16	Size. Size of the message data in bytes. Matches number of bytes in IPC_WBUF. 0x0
15:12	Sub Command ID (SUB_CMD_ID) This message ID is associated with multiple transactions; use this field to indicate the specific transaction. 0x0 – South IP allow gating 0x1 – South IP Ungate (Other Sub Command encodings are reserved)
11:9	Reserved
8	MSI. Generate MSI or not for the message. The state of this bit is left to the host driver—setting this bit allows the driver to be notified using interrupt on completion.
7:0	Command ID (CMD_ID) 0xE9



IPC_STS Register													
Bits	Description												
31:24	Reserved												
23:16	Error Code (ERROR CODE) . Indicates the Message-specific error code if ERROR is high. <table border="1"> <tr> <td>0 = none</td><td>No error</td></tr> <tr> <td>1 = command not supported</td><td>IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.</td></tr> <tr> <td>2 = command not serviced</td><td>IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).</td></tr> <tr> <td>3 = unable to service now</td><td>IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.</td></tr> <tr> <td>4 = command invalid</td><td>IPC_CMD.CMD_ID is invalid</td></tr> <tr> <td>5 = command failed</td><td>Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.</td></tr> </table>	0 = none	No error	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.	4 = command invalid	IPC_CMD.CMD_ID is invalid	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.
0 = none	No error												
1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.												
2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).												
3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.												
4 = command invalid	IPC_CMD.CMD_ID is invalid												
5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.												
15:8	Initiator ID (INIT_ID) . Hardware updates this value with the ID of the initiator.												
7:4	Command ID (CMD_ID) . Hardware copies value from SUB_CMD_ID field from IPC_CMD register.												
3	Reserved												
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1 to this bit to clear it and de-assert the interrupt.												
1	Error (ERROR) 0 = No error 1 = Error												
0	Ready/Busy (BUSY) 0 = IPC is ready to accept next IPC transaction 1 = IPC is busy												

6.5.2.4 PMC FW MSG Control

This message is used to read and write the PMC_FW_MSG_CTL register, IA FW could control PMC UART output, seven_seg output and message output verbosity level.

IPC_CMD Register	
Bits	Description
31:24	Reserved.
23:16	Size. Size of the message data in bytes. Matches number of bytes in IPC_WBUF. 0x0 – Read PMC FW_MSG_CTL register 0x1 – Write PMC FW_MSG_CTL register
15:12	Sub Command ID (SUB_CMD_ID) 0x0 – Read PMC FW_MSG_CTL register 0x1 – Write PMC FW_MSG_CTL register
11:9	Reserved
8	MSI. Generate MSI or not for the message. The state of this bit is left to the host driver—setting this bit allows the driver to be notified using interrupt on completion.
7:0	Command ID (CMD_ID) 0xEA

IPC_STS Register													
Bits	Description												
31:24	Reserved												
23:16	Error Code (ERROR CODE). Indicates the Message-specific error code if ERROR is high. <table border="1"> <tr> <td>0 = none</td><td>No error</td></tr> <tr> <td>1 = command not supported</td><td>IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.</td></tr> <tr> <td>2 = command not serviced</td><td>IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).</td></tr> <tr> <td>3 = unable to service now</td><td>IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.</td></tr> <tr> <td>4 = command invalid</td><td>IPC_CMD.CMD_ID is invalid</td></tr> <tr> <td>5 = command failed</td><td>Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out. The following error value will be returned in RBUF byte0</td></tr> </table>	0 = none	No error	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.	4 = command invalid	IPC_CMD.CMD_ID is invalid	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out. The following error value will be returned in RBUF byte0
0 = none	No error												
1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.												
2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).												
3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.												
4 = command invalid	IPC_CMD.CMD_ID is invalid												
5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out. The following error value will be returned in RBUF byte0												



IPC_STS Register	
Bits	Description
	bits[7:6, 3:0] = IC_TX_ABRT_SOURCE[7:6, 3:0] bit[4] = TIMEOUT bit[5] = bitwise_OR (IC_TX_ABRT_SOURCE[31:24, 16:8, 5:4])
15:8	Initiator ID (INIT_ID) . Hardware updates this value with the ID of the initiator.
7:4	Command ID (CMD_ID) . Hardware copies value from SUB_CMD_ID field from IPC_CMD register.
3	Reserved
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1 to this bit to clear it and de-assert the interrupt.
1	Error (ERROR) 0 = No error 1 = Error
0	Ready/Busy (BUSY) 0 = IPC is ready to accept next IPC transaction 1 = IPC is busy

IPC_STS Register									
Bits	Description								
31:24	Reserved								
23:16	Error Code (ERROR CODE) . Indicates the Message-specific error code if ERROR is high. <table border="1"> <tr> <td>0 = none</td><td>No error</td></tr> <tr> <td>1 = command not supported</td><td>IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.</td></tr> <tr> <td>2 = command not serviced</td><td>IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).</td></tr> <tr> <td>3 = unable to service now</td><td>IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.</td></tr> </table>	0 = none	No error	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.
0 = none	No error								
1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.								
2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).								
3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.								

IPC_STS Register		
Bits	Description	
	4 = command invalid	IPC_CMD.CMD_ID is invalid
	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out. The following error value will be returned in RBUF byte0 bits[7:6, 3:0] = IC_TX_ABRT_SOURCE[7:6, 3:0] bit[4] = TIMEOUT bit[5] = bitwise_OR (IC_TX_ABRT_SOURCE[31:24, 16:8, 5:4])
15:8	Initiator ID (INIT_ID) . Hardware updates this value with the ID of the initiator.	
7:4	Command ID (CMD_ID) . Hardware copies value from SUB_CMD_ID field from IPC_CMD register.	
3	Reserved	
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1 to this bit to clear it and de-assert the interrupt.	
1	Error (ERROR) 0 = No error 1 = Error	
0	Ready/Busy (BUSY) 0 = IPC is ready to accept next IPC transaction 1 = IPC is busy	

IPC_STS Register		
Bits	Description	
31:24	Reserved	
	Error Code (ERROR CODE) . Indicates the Message-specific error code if ERROR is high.	
23:16	0 = none	No error
	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.
	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).



IPC_STS Register		
Bits	Description	
	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.
	4 = command invalid	IPC_CMD.CMD_ID is invalid
	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out. The following error value will be returned in RBUF byte0 bits[7:6, 3:0] = IC_TX_ABRT_SOURCE[7:6, 3:0] bit[4] = TIMEOUT bit[5] = bitwise_OR (IC_TX_ABRT_SOURCE[31:24, 16:8, 5:4])
15:8	Initiator ID (INIT_ID) . Hardware updates this value with the ID of the initiator.	
7:4	Command ID (CMD_ID) . Hardware copies value from SUB_CMD_ID field from IPC_CMD register.	
3	Reserved	
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1 to this bit to clear it and de-assert the interrupt.	
1	Error (ERROR) 0 = No error 1 = Error	
0	Ready/Busy (BUSY) 0 = IPC is ready to accept next IPC transaction 1 = IPC is busy	

6.5.2.5 PM Debug

This message is used to config PM related setting, like LTR report, ignore

IPC_CMD Register		
Bits	Description	
31:24	Reserved.	

IPC_CMD Register									
Bits	Description								
23:16	<p>Size. Size of the message data in bytes. Matches number of bytes in IPC_WBUF.</p> <table border="1"> <tr> <td>SUB_CMD_ID</td><td>Size</td></tr> <tr> <td>0xD</td><td>1</td></tr> <tr> <td>0xE</td><td>0</td></tr> <tr> <td>0xF</td><td>2</td></tr> </table> <p>(Other Sub Command ID encodings are reserved)</p>	SUB_CMD_ID	Size	0xD	1	0xE	0	0xF	2
SUB_CMD_ID	Size								
0xD	1								
0xE	0								
0xF	2								
15:12	<p>Sub Command ID (SUB_CMD_ID) This message ID is associated with multiple transactions; use this field to indicate the specific transaction.</p> <p>0xD – LTR Report (IP to report is specified via IPC_WBUF) 0xE – LTR ignore read 0xF – LTR ignore write (Other Sub Command encodings are reserved)</p>								
11:9	Reserved								
8	MSI . Generate MSI or not for the message. The state of this bit is left to the host driver—setting this bit allows the driver to be notified using interrupt on completion.								
7:0	<p>Command ID (CMD_ID)</p> <p>0xEC</p>								

IPC_STS Register									
Bits	Description								
31:24	Reserved								
23:16	<p>Error Code (ERROR CODE). Indicates the Message-specific error code if ERROR is high.</p> <table border="1"> <tr> <td>0 = none</td><td>No error</td></tr> <tr> <td>1 = command not supported</td><td>IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.</td></tr> <tr> <td>2 = command not serviced</td><td>IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).</td></tr> <tr> <td>3 = unable to service now</td><td>IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.</td></tr> </table>	0 = none	No error	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.
0 = none	No error								
1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.								
2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).								
3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.								



IPC_STS Register		
Bits	Description	
	4 = command invalid	IPC_CMD.CMD_ID is invalid
	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.
15:8	Initiator ID (INIT_ID) . Hardware updates this value with the ID of the initiator.	
7:4	Command ID (CMD_ID) . Hardware copies value from IPC_DMD.SUB_CMD_ID	
3	Reserved	
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1 to this bit to clear it and de-assert the interrupt.	
1	Error (ERROR) 0 = No error 1 = Error	
0	Ready/Busy (BUSY) 0 = IPC is ready to accept next IPC transaction 1 = IPC is busy	

IPC_WBUF Register – when IPC_CMD.SUB_CMD_ID is 0xD (LTR Report)	
Byte	Description
15:1	NA for this SUB_CMD_ID
0	0x0 – Reserved for future LTR report usage 0x1 – LTR Report for AVS 0x2 – LTR Report for ISH 0x3 – LTR Report for XHCI 0x4 – LTR Report for LPSS 0x5 – LTR Report for MEX 0x6 – LTR Report for SCC 0x7-0xF Reserved for future LTR report usage

IPC_WBUF Register – when IPC_CMD.SUB_CMD_ID is 0XF (LTR Ignore Write)	
Byte	Description
15:2	NA for this IPC_CMD.SUB_CMD_ID
1:0	A 16-bit vector, with the following bit definitions 0:0 Reserved for future LTR ignore 1:1 AVS latency ignore 2:2 ISH latency ignore 3:3 XHCI latency ignore 4:4 LPSS latency ignore 5:5 MEX latency ignore 6:6 SCC latency ignore 7-15 Reserved for future LTR ignore

6.5.2.6 PHY Config

- BIOS will send this IPC1 messages after it decides how to handle the USB2, SSIC, and UFS (m) PHYs.
- BIOS will send sub-command 0x0 - PHY configuration is complete for the PHYs it has enabled and configured.
- BIOS will send sub-command 0x1 – PHY will not be configured for the PHYs which are not enabled. If a PHY is not enabled (for example UFS MPHYS), BIOS must have put the PHY into a low power state to allow S0ix and so on.

IPC_CMD Register	
Bits	Description
31:24	Reserved.
23:16	Size. Size of the message data in bytes. Matches number of bytes in IPC_WBUF required for the associated IPC_CMD.SUB_CMD_ID. 0x1
15:12	Sub Command ID (SUB_CMD_ID) This message ID is associated with multiple transactions; use this field to indicate the specific transaction. 0x0 – PHY configuration is complete 0x1 – PHY will not be configured (Other Sub Command encodings are reserved)
11:9	Reserved



8	MSI . Generate MSI or not for the message. The state of this bit is left to the host driver—setting this bit allows the driver to be notified using interrupt on completion.												
7:0	Command ID (CMD_ID) 0xEE												
IPC_STS Register													
Bits	Description												
31:24	Reserved												
23:16	Error Code (ERROR CODE) . Indicates the Message-specific error code if ERROR is high. <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">0 = none</td><td style="padding: 2px;">No error</td></tr> <tr> <td style="padding: 2px;">1 = command not supported</td><td style="padding: 2px;">IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.</td></tr> <tr> <td style="padding: 2px;">2 = command not serviced</td><td style="padding: 2px;">IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).</td></tr> <tr> <td style="padding: 2px;">3 = unable to service now</td><td style="padding: 2px;">IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.</td></tr> <tr> <td style="padding: 2px;">4 = command invalid</td><td style="padding: 2px;">IPC_CMD.CMD_ID is invalid</td></tr> <tr> <td style="padding: 2px;">5 = command failed</td><td style="padding: 2px;">Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.</td></tr> </table>	0 = none	No error	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.	4 = command invalid	IPC_CMD.CMD_ID is invalid	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.
0 = none	No error												
1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.												
2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).												
3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.												
4 = command invalid	IPC_CMD.CMD_ID is invalid												
5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.												
15:8	Initiator ID (INIT_ID) . Hardware updates this value with the ID of the initiator.												
7:4	Command ID (CMD_ID) . Hardware copies value from SUB_CMD_ID field from IPC_CMD register.												
3	Reserved												
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1 to this bit to clear it and de-assert the interrupt.												
1	Error (ERROR) 0 = No error 1 = Error												
0	Ready/Busy (BUSY) 0 = IPC is ready to accept next IPC transaction 1 = IPC is busy												

IPC_WBUF Register	
Byte	Description
15:1	NA for this IPC_CMD.CMD_ID
0	An 8 bit vector to specify which phy(s) the command applies to Bit[7:3] Reserved for future use Bit[2] USB2PHY Bit[1] UFSMPHY Bit[0] SSICMPHY

6.5.2.7 PMIC Access

IPC_CMD Register									
Bits	Description								
31:24	Reserved.								
23:16	Size. Size of the message data in bytes. Matches number of bytes in IPC_WBUF required for the associated IPC_CMD.SUB_CMD_ID. <table border="1"> <thead> <tr> <th>SUB_CMD_ID</th> <th>Size</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>2, 4, 6, 8, 10, 12, 14, 16 (1 read, 2 reads, ..., 8 reads)</td> </tr> <tr> <td>1</td> <td>3, 6, 9, 12, 15 (1 write, 2 writes, 3 writes, 4 writes, 5 writes)</td> </tr> <tr> <td>2</td> <td>4, 8, 12, 16 (1 rmw, 2 rmw, 3 rmw, 4 rmw)</td> </tr> </tbody> </table> (Other Sub Command encodings are reserved)	SUB_CMD_ID	Size	0	2, 4, 6, 8, 10, 12, 14, 16 (1 read, 2 reads, ..., 8 reads)	1	3, 6, 9, 12, 15 (1 write, 2 writes, 3 writes, 4 writes, 5 writes)	2	4, 8, 12, 16 (1 rmw, 2 rmw, 3 rmw, 4 rmw)
SUB_CMD_ID	Size								
0	2, 4, 6, 8, 10, 12, 14, 16 (1 read, 2 reads, ..., 8 reads)								
1	3, 6, 9, 12, 15 (1 write, 2 writes, 3 writes, 4 writes, 5 writes)								
2	4, 8, 12, 16 (1 rmw, 2 rmw, 3 rmw, 4 rmw)								
15:12	Sub Command ID (SUB_CMD_ID) This message ID is associated with multiple transactions; use this field to indicate the specific transaction. 0x0 – Read (number of registers to read is specified by IPC_CMD.SIZE) 0x1 – Write (number of registers to write is specified by IPC_CMD.SIZE) 0x2 – Read-Modify-Write (number of registers to read-modify-write is specified by IPC_CMD.SIZE) (Other Sub Command encodings are reserved)								
11:9	Reserved								
8	MSI. Generate MSI or not for the message. The state of this bit is left to the host driver—setting this bit allows the driver to be notified using interrupt on completion.								
7:0	Command ID (CMD_ID) 0xFF								



IPC_CMD Register													
Bits	Description												
IPC_STS Register													
Bits	Description												
31:24	Reserved												
23:16	<p>Error Code (ERROR_CODE). Indicates the Message-specific error code if ERROR is high.</p> <table border="1"> <tr> <td>0 = none</td><td>No error</td></tr> <tr> <td>1 = command not supported</td><td>IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.</td></tr> <tr> <td>2 = command not serviced</td><td>IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).</td></tr> <tr> <td>3 = unable to service now</td><td>IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.</td></tr> <tr> <td>4 = command invalid</td><td>IPC_CMD.CMD_ID is invalid</td></tr> <tr> <td>5 = command failed</td><td>Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out. The following error value will be returned in RBUF byte0 bits[7:6, 3:0] = IC_TX_ABRT_SOURCE[7:6, 3:0] bit[4] = TIMEOUT bit[5] = bitwise_OR (IC_TX_ABRT_SOURCE[31:24, 16:8, 5:4])</td></tr> </table>	0 = none	No error	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.	4 = command invalid	IPC_CMD.CMD_ID is invalid	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out. The following error value will be returned in RBUF byte0 bits[7:6, 3:0] = IC_TX_ABRT_SOURCE[7:6, 3:0] bit[4] = TIMEOUT bit[5] = bitwise_OR (IC_TX_ABRT_SOURCE[31:24, 16:8, 5:4])
0 = none	No error												
1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.												
2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).												
3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.												
4 = command invalid	IPC_CMD.CMD_ID is invalid												
5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out. The following error value will be returned in RBUF byte0 bits[7:6, 3:0] = IC_TX_ABRT_SOURCE[7:6, 3:0] bit[4] = TIMEOUT bit[5] = bitwise_OR (IC_TX_ABRT_SOURCE[31:24, 16:8, 5:4])												
15:8	Initiator ID (INIT_ID) . Hardware updates this value with the ID of the initiator.												
7:4	Command ID (CMD_ID) . Hardware copies value from SUB_CMD_ID field from IPC_CMD register.												
3	Reserved												
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1 to this bit to clear it and de-assert the interrupt.												
1	Error (ERROR) 0 = No error 1 = Error												
0	Ready/Busy (BUSY) 0 = IPC is ready to accept next IPC transaction												

IPC_CMD Register	
Bits	Description
...	IPC is busy

IPC_WBUF Register – when IPC_CMD.SUB_CMD_ID is 0X0 (PMIC Read)	
Byte	Description
15:14	8 reads: address (base+offset) of 8 th register to be read
13:12	7 reads: address (base+offset) of 7 th register to be read
11:10	6 reads: address (base+offset) of 6 th register to be read
9:8	5 reads: address (base+offset) of 5 th register to be read
7:6	4 reads: address (base+offset) of 4 th register to be read
5:4	3 reads: address (base+offset) of 3 rd register to be read
3:2	2 reads: address (base+offset) of 2nd register to be read
1:0	1 read: address (base+offset) of register to be read

IPC_WBUF Register – when IPC_CMD.SUB_CMD_ID is 0X1 (PMIC Write)	
Byte	Description
	<p>1 write: IPC_WBUF[1:0] = address, IPC_WBUF[2] = data</p> <p>2 writes: IPC_WBUF[1:0] = address, IPC_WBUF[4] = data IPC_WBUF[3:2] = address, IPC_WBUF[5] = data</p> <p>3 writes: IPC_WBUF[1:0] = address, IPC_WBUF[6] = data IPC_WBUF[3:2] = address, IPC_WBUF[7] = data IPC_WBUF[5:4] = address, IPC_WBUF[8] = data</p> <p>4 writes: IPC_WBUF[1:0] = address, IPC_WBUF[8] = data</p>



	<p>IPC_WBUF[3:2] = address, IPC_WBUF[9] = data IPC_WBUF[5:4] = address, IPC_WBUF[10] = data IPC_WBUF[7:6] = address, IPC_WBUF[11] = data</p> <p>5 writes:</p> <p>IPC_WBUF[1:0] = address, IPC_WBUF[10] = data IPC_WBUF[3:2] = address, IPC_WBUF[11] = data IPC_WBUF[5:4] = address, IPC_WBUF[12] = data IPC_WBUF[7:6] = address, IPC_WBUF[13] = data IPC_WBUF[9:8] = address, IPC_WBUF[14] = data</p>
IPC_WBUF Register – when IPC_CMD.SUB_CMD_ID is 0X2 (PMIC R-M-W)	
Byte	Description
	<p>1 read-modify-write: IPC_WBUF[1:0] = address, IPC_WBUF[2] = data, IPC_WBUF[3] = mask</p> <p>2 read-modify-writes: IPC_WBUF[1:0] = address, IPC_WBUF[4] = data, IPC_WBUF[6] = mask IPC_WBUF[3:2] = address, IPC_WBUF[5] = data, IPC_WBUF[7] = mask</p> <p>3 read-modify-writes: IPC_WBUF[1:0] = address, IPC_WBUF[6] = data, IPC_WBUF[9] = mask IPC_WBUF[3:2] = address, IPC_WBUF[7] = data, IPC_WBUF[10] = mask IPC_WBUF[5:4] = address, IPC_WBUF[8] = data, IPC_WBUF[11] = mask</p> <p>4 read-modify-writes: IPC_WBUF[1:0] = address, IPC_WBUF[8] = data, IPC_WBUF[12] = mask IPC_WBUF[3:2] = address, IPC_WBUF[9] = data, IPC_WBUF[13] = mask IPC_WBUF[5:4] = address, IPC_WBUF[10] = data, IPC_WBUF[14] = mask IPC_WBUF[7:6] = address, IPC_WBUF[11] = data, IPC_WBUF[15] = mask</p>

	IPC_RBUF Register - when IPC_CMD.SUB_CMD_ID is 0X0 (PMIC Read)
Byte	Description
15:8	NA for this IPC_CMD.SUB_CMD_ID
7	8 reads: data from 8 th register to be read
6	7 reads: data from 7 th register to be read
5	6 reads: data from 6 th register to be read

4	5 reads: data from 5 th register to be read
3	4 reads: data from 4 th register to be read
2	3 reads: data from 3 rd register to be read
1	2 reads: data from 2nd register to be read
0	1 read: data from register to be read

IPC_RBUF Register - when IPC_CMD.SUB_CMD_ID is 0X1 (PMIC Write)	
Byte	Description
15:0	NA for this IPC_CMD.SUB_CMD_ID
IPC_RBUF Register - when IPC_CMD.SUB_CMD_ID is 0X2 (PMIC R-M-W)	
Byte	Description
1	NA for this IPC_CMD.SUB_CMD_ID
3	4 read-modify-writes: result of RMW to 4 th register
2	3 read-modify-writes: result of RMW to 3 rd register
1	2 read-modify-writes: result of RMW to 2 nd register
0	1 read-modify-writes: result of RMW to 1 st register

6.5.2.8 Blacklist Select

IA FW (via secure boot, and other information it knows) is making the determination of Trusted vs. Untrusted OS and Trusted vs Untrusted ISH. IA FW will send a "ONE-TIME" IPC1 message to tell PMC firmware to switch from the Initial blacklists to the Runtime blacklists. There are 4 possible combinations:

- IA Insecure, ISH Insecure
- IA Insecure, ISH Secure
- IA Secure, ISH Insecure
- IA Secure, ISH Secure (NOTE: likely will not be used)



IA FW will send the message after "end of post" message - SAI has changed by this time to HOST_SAI_UNTRUSTED. IA FW will indicate in the IPC_CMD.SUB_CMD_ID which combination of runtime blacklists based on the IA FW determination.

PMC firmware, on receipt of the IPC1 message, will immediately switch to the requested IA Runtime blacklist and ISH Runtime blacklist. From that point, PMC FW will reject any additional requests to change the runtime blacklists. (the change is permanent until the next reset, and until that time any additional IPC1 request to update will be rejected).

IPC_CMD Register	
	Description
Reserved.	
Size. Size of the message data in bytes. Matches number of bytes in IPC_WBUF. 0x00	
Sub Command ID (SUB_CMD_ID) 0x0 – Use IA Insecure Blacklist, Use ISH Insecure Blacklist to filter PMIC Accesses 0x1 – Use IA Insecure Blacklist, Use ISH Secure Blacklist to filter PMIC Accesses 0x2 – Use IA Secure Blacklist, Use ISH Insecure Blacklist to filter PMIC Accesses 0x3 – Use IA Secure Blacklist, Use ISH Secure Blacklist to filter PMIC Accesses	
Reserved	
MSI. Generate MSI or not for the message. The state of this bit is left to the host driver—setting this bit allows the driver to be notified using interrupt on completion.	
Command ID (CMD_ID) 0xEF	

IPC_STS Register					
Bits	Description				
31:24	Reserved				
23:16	Error Code (ERROR CODE). Indicates the Message-specific error code if ERROR is high. <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">0 = none</td> <td style="padding: 2px;">No error</td> </tr> <tr> <td style="padding: 2px;">1 = command not supported</td> <td style="padding: 2px;">IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.</td> </tr> </table>	0 = none	No error	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.
0 = none	No error				
1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.				

IPC_STS Register		
Bits	Description	
	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).
	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.
	4 = command invalid	IPC_CMD.CMD_ID is invalid
	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.
15:8	Initiator ID (INIT_ID) . Hardware updates this value with the ID of the initiator.	
7:4	Command ID (CMD_ID) . Hardware copies value from SUB_CMD_ID field from IPC_CMD register.	
3	Reserved	
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1 to this bit to clear it and de-assert the interrupt.	
1	Error (ERROR) 0 = No error 1 = Error	
0	Ready/Busy (BUSY) 0 = IPC is ready to accept next IPC transaction 1 = IPC is busy	

6.5.2.9 USB 3.3V Control

IPC_CMD Register		
Bits	Description	
31:24	Reserved.	
23:16	Size . Size of the message data in bytes. Matches number of bytes in IPC_WBUF. 0x00	
15:12	Sub Command ID (SUB_CMD_ID) 0x0 – Turn off USB 3.3V 0x1 – Turn on USB 3.3V	



IPC_CMD Register													
Bits	Description												
11:9	Reserved												
8	MSI . Generate MSI or not for the message. The state of this bit is left to the host driver—setting this bit allows the driver to be notified using interrupt on completion.												
7:0	Command ID (CMD_ID) 0xFO												
IPC_STS Register													
Bits	Description												
31:24	Reserved												
23:16	Error Code (ERROR CODE) . Indicates the Message-specific error code if ERROR is high. <table border="1"> <tr> <td>0 = none</td><td>No error</td></tr> <tr> <td>1 = command not supported</td><td>IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.</td></tr> <tr> <td>2 = command not serviced</td><td>IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).</td></tr> <tr> <td>3 = unable to service now</td><td>IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.</td></tr> <tr> <td>4 = command invalid</td><td>IPC_CMD.CMD_ID is invalid</td></tr> <tr> <td>5 = command failed</td><td>Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out. The following error value will be returned in RBUF byte0 bits[7:6, 3:0] = IC_TX_ABRT_SOURCE[7:6, 3:0] bit[4] = TIMEOUT bit[5] = bitwise_OR (IC_TX_ABRT_SOURCE[31:24, 16:8, 5:4]) </td></tr> </table>	0 = none	No error	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.	4 = command invalid	IPC_CMD.CMD_ID is invalid	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out. The following error value will be returned in RBUF byte0 bits[7:6, 3:0] = IC_TX_ABRT_SOURCE[7:6, 3:0] bit[4] = TIMEOUT bit[5] = bitwise_OR (IC_TX_ABRT_SOURCE[31:24, 16:8, 5:4])
0 = none	No error												
1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.												
2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).												
3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.												
4 = command invalid	IPC_CMD.CMD_ID is invalid												
5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out. The following error value will be returned in RBUF byte0 bits[7:6, 3:0] = IC_TX_ABRT_SOURCE[7:6, 3:0] bit[4] = TIMEOUT bit[5] = bitwise_OR (IC_TX_ABRT_SOURCE[31:24, 16:8, 5:4])												
15:8	Initiator ID (INIT_ID) . Hardware updates this value with the ID of the initiator.												
7:4	Command ID (CMD_ID) . Hardware copies value from SUB_CMD_ID field from IPC_CMD register.												
3	Reserved												

IPC_CMD Register	
Bits	Description
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1 to this bit to clear it and de-assert the interrupt.
1	Error (ERROR) 0 = No error 1 = Error
0	Ready/Busy (BUSY) 0 = IPC is ready to accept next IPC transaction 1 = IPC is busy
IPC_RBUF Register – on IPC_STS.ERROR_CODE = "Command Failed"	
Byte	Description
15:1	0
0	bits[7:6, 3:0] = IC_TX_ABRT_SOURCE[7:6, 3:0] bit[4] = TIMEOUT bit[5] = bitwise_OR (IC_TX_ABRT_SOURCE[31:24, 16:8, 5:4]

6.5.2.10 North Peak control

IPC_CMD Register	
Bits	Description
31:24	Reserved.
23:16	Size . Size of the message data in bytes. Matches number of bytes in IPC_WBUF. 0x0
15:12	Sub Command ID (SUB_CMD_ID) This message ID is associated with multiple transactions; use this field to indicate the specific transaction. 0x0 – Disable North Peak 0x1 – Enable North Peak (Other Sub Command encodings are reserved)
11:9	Reserved



8	MSI . Generate MSI or not for the message. The state of this bit is left to the host driver—setting this bit allows the driver to be notified using interrupt on completion.												
7:0	Command ID (CMD_ID) 0xED												
IPC_STS Register													
Bits	Description												
31:24	Reserved												
23:16	<p>Error Code (ERROR CODE). Indicates the Message-specific error code if ERROR is high.</p> <table border="1"> <tbody> <tr> <td>0 = none</td><td>No error</td></tr> <tr> <td>1 = command not supported</td><td>IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.</td></tr> <tr> <td>2 = command not serviced</td><td>IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).</td></tr> <tr> <td>3 = unable to service now</td><td>IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.</td></tr> <tr> <td>4 = command invalid</td><td>IPC_CMD.CMD_ID is invalid</td></tr> <tr> <td>5 = command failed</td><td>Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.</td></tr> </tbody> </table>	0 = none	No error	1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.	2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).	3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.	4 = command invalid	IPC_CMD.CMD_ID is invalid	5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.
0 = none	No error												
1 = command not supported	IPC_CMD.CMD_ID is correct, but there is a problem in formatting of the other IPC_CMD fields. Example: wrong IPC_CMD.SIZE.												
2 = command not serviced	IPC_CMD is correct, but service you asked for is permanently disabled (fused off, blacklisted, etc).												
3 = unable to service now	IPC_CMD is correct, but PMC cannot service at this time. Example: command which can only be called once, or can only happen at a particular time.												
4 = command invalid	IPC_CMD.CMD_ID is invalid												
5 = command failed	Attempted to service the command, but it failed for some reason! Example: pmic access failed/timed out.												
15:8	Initiator ID (INIT_ID) . Hardware updates this value with the ID of the initiator.												
7:4	Command ID (CMD_ID) . Hardware copies value from SUB_CMD_ID field from IPC_CMD register.												
3	Reserved												
2	IRQ (IRQ) . Hardware sets this bit based on FW completion of a message which specified IPC_CMD.MSI. Causes an interrupt to be asserted. Host is required to write '1 to this bit to clear it and de-assert the interrupt.												
1	Error (ERROR) 0 = No error : Error												
0	Ready/Busy (BUSY) 0 = IPC is ready to accept next IPC transaction : IPC is busy												



6.5.3 ACPI Requirement

IA FW should report IPC1 as an ACPI device, report the MMIO address space and interrupt resources via _crs and IPC1 driver on host would claim the device address space.

Table 6-2. IPC1 ACPI ASL Sample Code

```
Device(IPC1)
{
    Name (_ADR, 0)
    Name (_HID, "80860XXX")
    Name (_CID, "80860XXX")
    Name (_DDN, "Intel(R) IPCL controller ")
    Name (_UID, 1)
    Name (RBUF, ResourceTemplate ())
    {
        Memory32Fixed (ReadWrite, 0x00000000, 0x00001000, BAR0)
        Interrupt (ResourceConsumer, Level, ActiveLow, Exclusive, , , )
{xx} // IPC1 IRQ }
        Method (_CRS, 0x0, NotSerialized)
    {
        CreateDwordField(^RBUF, ^BAR0._BAS, B0BA)
        CreateDwordField(^RBUF, ^BAR0._LEN, B0LN)
        Store(D13A, B0BA) // D13A is the BAR high address for B0/D13/F1
        Store(D13L, B0LN) //D13L is the BAR low address for B0/D13/F1
    Return (RBUF)
    }
    Method (_STA, 0x0, NotSerialized)
    {
        Return (0xF)
    }
}
```

6.6 Additional Power Management Programming

BIOS requires to program the registers listed below for S0ix enable:

- Set Bit17 and Bit16 in PMC MMIO offset 0x1048 [17:15] if S0ix is enabled

BIOS are expected to change the setup option of PCIE Root Port from "Enable" to "Auto". Setting to "Auto" will set the function disable bit of the port if no PCIE card is detected. BIOS needs to issue cold reset to do PMC function disable. Thus, the auto reboot during boot up is expected behavior.

7 Image Processing Unit (IPU)

The IPU aka I-unit in the Apollo Lake (APL) SoC consists of the Processing Subsystem (PS), which is an advanced Image Signal processor (ISP), and the Input Subsystem (IS), which contains the MIPI CSI2 controllers. Also, Apollo Lake ISP has higher performance than previous generation platforms (e.g. Cherry Trail).

7.1

Enable/Disable IPU Device

1. IA FW could further configure IPU only if it is enabled. This can be verified by reading B0.D0.F0.R0E8h [31] – IPU_Disabled. If this bit is 0, the IPU is enabled.
2. IA FW may then proceed to enable the device by setting B0.D0.F0.R54h [5].
3. If IA FW wishes to disable the IPU device, it must clear B0.D0.F0.R54h [5] and not proceed with any other steps are described in below sections.

7.2

Setup and Lock IMGU Device's Config Space Registers

SVID_SID - Subsystem Vendor ID and Subsystem ID Register. IA FW must setup B0.D0.F3.R2Ch bits [31:16] and [15:0] to a value of 0x0000. This operation locks this register.

IA firmware also needs to configure the interrupt routine mode for both APIC and PIC in ACPI level.

7.3

CSI Lane Configuration

All CSI-2 C-PHY and D-PHY lanes are configured by TXE. IA FW has no involvement here.

7.3.1

FLIS Configuration

The following FLIS registers need to be configured by IA FW through sideband access (SBREG_BAR)

1. CSI_CLKTRIM
2. CSI_DATATTRIM
3. CSI_DATATTRIM1
4. CSI_CTLE
5. CSI_RCOMP_CONTROL

To access each of the above registers, IA FW must execute below sequence:

1. Poll P2SB(Primary to Sideband Bridge) Offset D8h[0] = 0b



2. Write P2SB Offset D0h[15:0] = address offset of the FLIS register to be written, and P2SB Offset D0h[31:24] = Destination Port ID of the FLIS IP (0xAA).
3. Write P2SB Offset DAh[15:0] = F000h
4. Write P2SB Offset D8h[15:8] = 07h (write op-code)
5. Write P2SB Offset D4h[31:0] with the FLIS register data that should be written
6. Write P2SB Offset DAh[15:0] = F000h
7. Set P2SB Offset D8h[0] = 1b
8. Poll P2SB Offset D8h[0] = 0b
9. Check if P2SB Offset D8h[2:1] = 00b for successful transaction

7.4 GPIO Allocation for IPU

There are 12 GPIOs that can be configured directly by IPU. IA FW should allocate the required GPIOs to the IPU. One of the GPIOs has a pre-defined use case as Flash trigger.

7.5 AVStream Virtual Device Enumeration

Baseline platforms would have integrated GFX enabled. For such systems, the AVStream virtual device should be enumerated as a child of the GFX device. [Section 6.5.1](#) describes this requirement.

There might be platforms that the integrated GFX is fused out. In such case the AVStream virtual device should be enumerated by the IA FW as ACPI device that is not child of GFX. [Section 6.5.2](#) describes this requirement.

7.5.1 Enumeration of AVStream Virtual Device as Child of GFX

IA FW should list the IPU's AVStream virtual Camera device as GPU child device with a unique ID 0x0002wwww in the GFX _DOD method, where "wwww", the latter 16 bits of the ACPI id, is IPU's AVStream virtual Camera's unique ID within the scope of the GFX driver.

IA FW shall place the IPU's AVStream virtual Camera device as a child device under GFX ACPI name space. The Camera device definition should have a matching _ADR with same unique ID - "wwww" values defined in the GFX _DOD method.

Intel GFX driver will recognize the "0x0002CA00" as a unique ID for IPU's AVStream virtual Camera device by finding out that bit 17 of "TBD" is set, and therefore Intel GFX driver will enumerate as its child the IPU's AVStream virtual Camera device by reporting IPU AVStream virtual Camera device's device HardwareID TBD, selected by the GFX driver based on the unique "TBD" value to the display port driver that will match IPU's AVStream INF file.

At the same time, in order to be successfully loaded by PnP manager, IPU's AVStream driver should also specify a HardwareID (TBD) in its INF file. This HardwareID is used by the GFX driver to specify to the display port driver which driver to load.

As part of the ACPI name space for the AVStream child device IA FW shall provide the following details:

- a. _ADR with ACPI ID that matches the ACPI ID that appears in the _DOD method in the GFX device definition.
- b. No MMIO resources (the virtual device does not consume memory map)
- c. Child of GFX driver flag is TRUE

Sample code:

```

Device(GPU0)

{
    Name(_ADR, 0x00020000) // GFX is a PCI device - Dev2,Function0

    Name (_DOD,           // Identifies the children of this Graphics device.
          // Each integer must be unique within the GPU0 namespace
          Package()
    {
        0x0002xxxx, // 0xXXXX is the ID for APL camera AVStream
                    // virtual driver. This number should match the one
                    // in _ADR in the SKC0 definition.
        // 0x0002 - means, it is a Non-VGA Device, cannot be
        // detected by the VGA BIOS, and uses a vendor
        // specific ID format in bits 15:0
        //...Other child device IDs
    }
}

// Other required objects for the GPU0 device ...

// Device IPU0 is the Camera AVStream virtual device and it appears under
GPU0

Device(IPU0)           // Camera AVStream virtual device name
{
    Name(_ADR, 0x0000XXXX) // The identifier for this device (Same as in

```



```
// _DOD above). This is required so GFX driver  
can  
  
// associate a matching device ID for the  
AVStream  
  
// driver and provide it to PnP (this device ID  
// should appear in the INF file of the AVStream  
// driver).  
  
// The following is a technique that may be used (per OEM needs) to  
prevent  
  
// the load of the camera device in one of the following cases:  
  
// - Camera device are fused out  
  
// - If the platform setup requires that in a secured boot the camera  
device  
  
// should not be enabled  
  
Method (_STA, 0, NotSerialized) {  
  
    If(LEqual(ISPD,1)){ // Dev2 need report ISP0 as GFX0 child  
  
        Return (0xF)  
  
    }  
  
    Else{ // Dev2 should NOT report ISP0 as GFX0 child  
  
        Return (0x0)  
  
    }  
  
}  
  
} // End SKC0  
  
} // End GPU0
```

7.5.2 Enumeration of AVStream Virtual Device as ACPI Device

In the absence of integrated GFX, BIOS shall enumerate the IPU AVStream virtual Camera controller device as ACPI device. In this case, IA FW should provide to this device driver ‘resources’ that will flag the driver to know that it should not attempt to connect to a GFX driver as its parent, as there is no such driver expected. As a result, the IPU AVStream driver would not make use of VRAM allocation in those PINs that could use VRAM in the presence of Intel GFX driver.

IA FW enumeration should report a proper HW Device ID that should match with Device ID mentioned in the IPU AVStream driver INF file).

Based on the HW Device ID reported, OS PNP manager will load the IPU AVStream Driver that is specified in the INF file.

As part of the ACPI enumeration IA FW shall provide the following details of the virtual device:

- a. ACPI HID/CID (this shall match the DeviceID that appears in IPU AVStream driver's INF file)
- b. No MMIO (the virtual device does not consume memory map)
- c. Child of GFX driver flag is FALSE

Example ASL code:

```

// Device IPU0 is the Camera AVStream virtual device and it appears under _SB
Device(IPU0)           // Camera AVStream virtual device name
{
    Name (_HID, "TBD") // This is the ACPI ID of Camera AVStream driver
    If (Windows OS version is less than Windows 8)
        Name (_CID, "PNP0C02") // Allows OS to not yellow bang if Intel do not
                                // provide driver for that OS version, otherwise
                                // if we have driver for every OS, then no need
                                // for _CID at all.

    // The following is a technique that may be used (per OEM needs) to
    prevent
    // the load of the camera device in one of the following cases:
    // - Camera device are fused out
    // - If the platform setup requires that in a secured boot the camera
    device
    // should not be enabled

    Method (_STA, 0, NotSerialized) {
        If(LEqual(ISPD,1)){ // ACPI need report ISP0 as available device
            Return (0xF)
        }
        Else{               // ACPI should NOT report ISP0 as available
device
            Return (0x0)
        }
    }
}

```



```
    }  
}  
  
}
```

7.6 ACPI Requirement for IA FW

7.6.1 Device Definition

BIOS should expose all the camera devices connected on the platform along with the FLASH LED resources as ACPI device. For each camera on the platform, ACPI need to provide _HID/_CID/_DDN/_STA/UID interfaces as well.

Example for rear camera IMX214 in RVP:

Category	ACPI Items	Requirement	Comment						
Generic	Device Name	_SB.PCI0.I2C2.CAM0	It should be under the I2C controller node.						
	_ADR	Zero	No specific requirement.						
	_HID	SONY214A	The sensor hardware identifier, provided by Sensor Vendor. <table border="1"><tr><td>CMOS Sensor</td><td>Hardware ID</td></tr><tr><td>IMX214</td><td>SONY214A</td></tr><tr><td>IMX132</td><td>SONY132A</td></tr></table>	CMOS Sensor	Hardware ID	IMX214	SONY214A	IMX132	SONY132A
CMOS Sensor	Hardware ID								
IMX214	SONY214A								
IMX132	SONY132A								
_CID	Same as _HID	Same as _HID.							
_UID	ONE	No specific requirement.							
	_DDN	SONY IMX214	Device Name should be defined as sensor OEM company and sensor name. <table border="1"><tr><td>CMOS sensor</td><td>Device Name</td></tr><tr><td>IMX214</td><td>"SONY IMX214"</td></tr><tr><td>IMX132</td><td>"SONY IMX132"</td></tr></table>	CMOS sensor	Device Name	IMX214	"SONY IMX214"	IMX132	"SONY IMX132"
CMOS sensor	Device Name								
IMX214	"SONY IMX214"								
IMX132	"SONY IMX132"								

Category	ACPI Items	Requirement	Comment
	_STA	0xF	No specific requirement.
Resource list (_CRS)	I2C for CMOS	I2cSerialBus(0x001A, ControllerInitiated, 0x00061A80, AddressingMode7Bit, "_SB.PCI0.I2C2", 0x00, ResourceConsumer,,)	The I2C used by COMS sensor. The I2C slave address. Depends the sensor module. IMX214 Slave address: 0x1A IMX132 Slave address: 0x36
	I2C for VCM	I2cSerialBus(0x000C, ControllerInitiated, 0x00061A80, AddressingMode7Bit, "_SB.PCI0.I2C2", 0x00, ResourceConsumer,,)	The I2C used by VCM. It would be used when the sensor module had a VCM. The I2C slave address depends on the sensor module.
	I2C for EEPROM	I2cSerialBus(0x0050, ControllerInitiated, 0x00061A80, AddressingMode7Bit, "_SB.PCI0.I2C2", 0x00, ResourceConsumer,,)	The I2C used by EEPROM. It would be used when the sensor module had an EEPROM. The I2C slave address depends on the sensor module.
	Reset Pin GPIO	GpioIo(Exclusive, PullDefault, 0x0000, 0x0000, IoRestrictionOutputOnly, "_SB.GPO0", 0x00, ResourceConsumer,,) { // Pin list 0x0041 }	GPIO_65: CAM0_RST_N
	_PLD	{ /* 0000 */ 0x82, 0x00, 0x00, 0x00, 0x00, /* 0008 */ 0x69, 0x0C, 0x00, 0x00, 0x00, /* 0010 */ 0xFF, 0xFF, 0xFF, 0xFF } { /* 0000 */ 0x82, 0x00, 0x00, 0x00, 0x00, /* 0008 */ 0x61, 0x0C, 0x00, 0x00, 0x00, /* 0010 */ 0xFF, 0xFF, 0xFF, 0xFF }	The PLD is corresponding of the platform board itself including the rear or front information. In Intel RVP, it is defined as: Rear Sensor: { /* 0000 */ 0x82, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0008 */ 0x69, 0x0C, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x00, /* 0010 */ 0xFF, 0xFF, 0xFF, 0xFF } Front Sensor: { /* 0000 */ 0x82, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0008 */ 0x61, 0x0C, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x00, /* 0010 */ 0xFF, 0xFF, 0xFF, 0xFF }



Category	ACPI Items	Requirement	Comment
	SSDB	Method is for camera sensor. Return buffer within detail settings.	<pre>Method (SSDB) { Name(PAR, Buffer(0x6C) { 0x00, //Version 0x70, //CRD board Type, 0: UNKNOWN 0x20: CRD-D 0x30: CRD-G 0x40: PPV 0x50: CRD-G2 0x70: CRD_G_BXT 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, //GUID for CSI2 host controller 0x00, //DevFunction 0x00, //Bus 0x00, 0x00, 0x00, 0x00, //DphyLinkEnFuses 0x00, 0x00, 0x00, 0x00, //ClockDiv 0x00, //LinkUsed 0x04, //LaneUsed 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_TERMEN_CLANE 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_SETTLE_CLANE 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_TERMEN_DLNE0 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_SETTLE_DLNE0 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_TERMEN_DLNE1 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_SETTLE_DLNE1 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_TERMEN_DLNE2 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_SETTLE_DLNE2 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_TERMEN_DLNE3 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_SETTLE_DLNE3 0x00, 0x00, 0x00, 0x00, //MaxLaneSpeed</pre>

Category	ACPI Items	Requirement	Comment
			<pre> 0x00, //SensorCalibrationFileIdx 0x00, 0x00, 0x00, //SensorCalibrationFileIdxInMBZ 0x00, //RomType: None 0x00, //VcmType: None 0x08, //Platform info 0x00, //Platform sub info 0x02, //Flash Disabled 0x01, //Privacy LED supported 0x00, //0 degree 0x01, //MIPI link/lane number defined in ACPI 0x00, 0xF8, 0x24, 0x01, // MCLK: 19200000Hz 0x03, //Control logic ID 0x00, 0x00, 0x00, 0x01, // M_CLK 0x00, 0x00, 0x00, //Reserved 0x00, 0x00, 0x00, 0x00, 0x00, //Reserved 0x00, 0x00, 0x00, 0x00, 0x00, //Reserved }) Return (PAR) } </pre>
	CLDB	Method is for control logic.	<pre> Method(CLDB) { Name(PAR, Buffer(0x20) { 0x00, //Version 0x01, //Control logic Type 0: UNKNOWN 1: DISCRETE 2: PMIC TPS68470 3: PMIC uP6641 </pre>



Category	ACPI Items	Requirement	Comment
			0x02, //Control logic ID: Control Logic 2 0x70, //CRD board Type, 0: UNKNOWN 0x20: CRD-D 0x30: CRD-G 0x40: PPV 0x50: CRD-G2 0x70: CRD_G_BXT 0x00, 0x00, 0x00, 0x00, // Reserved 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // Reserved 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // Reserved 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // Reserved }) Return (PAR) }

Example for rear camera IMX135 in RVP:

Category	ACPI Items	Requirement
Generic	Device Name	_SB.PCI0.I2C2.CAM1
	_ADR	Zero
	_HID	INT3471
	_CID	Same as _HID
	_UID	ONE
	_DDN	SONY IMX135
	_STA	0xF
Resource list (_CRS)	I2C for CMOS	I2cSerialBus(0x0010,ControllerInitiated, 0x00061A80,AddressingMode7Bit, "_SB.PCI0.I2C2",0x00,ResourceConsumer,,)
	I2C for VCM	I2cSerialBus(0x000E,ControllerInitiated, 0x00061A80,AddressingMode7Bit, "_SB.PCI0.I2C2",0x00,ResourceConsumer,,)
	I2C for EEPROM	
	_PLD	{ /* 0000 */ 0x82, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0008 */ 0x69, 0x0C, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00,

		/* 0010 */ 0xFF, 0xFF, 0xFF, 0xFF }
	SSDB	0x00, //Version 0x70, //SKU: CRD_G_BXT 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, //GUID for CSI2 host controller 0x00, //DevFunction 0x00, //Bus 0x00, 0x00, 0x00, 0x00, //DphyLinkEnFuses 0x00, 0x00, 0x00, 0x00, //ClockDiv 0x06, //LinkUsed 0x04, //LaneUsed 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_TERMEN_CLANE 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_SETTLE_CLANE 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_TERMEN_DLNEO 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_SETTLE_DLNEO 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_TERMEN_DLNE1 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_SETTLE_DLNE1 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_TERMEN_DLNE2 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_SETTLE_DLNE2 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_TERMEN_DLNE3 0x00, 0x00, 0x00, 0x00, //CSI_RX_DLY_CNT_SETTLE_DLNE3 0x00, 0x00, 0x00, 0x00, //MaxLaneSpeed 0x00, //SensorCalibrationFileIdx 0x00, 0x00, 0x00, //SensorCalibrationFileIdxInMBZ 0x00, //RomType: NONE 0x03, //VcmType: AD5816 0x08, //Platform info BXT 0x00, //Platform sub info 0x03, //Flash enabled 0x00, //Privacy LED not supported 0x00, //0 degree 0x01, //MIPI link/lane defined in ACPI 0x00, 0xF8, 0x24, 0x01, //MCLK: 19200000Hz 0x00, //Control logic ID 0x00, 0x00, 0x00, 0x02, //M_CLK 0x00, 0x00, 0x00, //Reserved



		0x00, 0x00, 0x00, 0x00, 0x00, //Reserved 0x00, 0x00, 0x00, 0x00, //Reserved
--	--	--

Example for front camera IMX132 in RVP,

Category	ACPI Items	Requirement	Comment					
Generic	Device Name	_SB.PCIO.I2C3.CAM1	It should be under the I2C controller node.					
	_ADR	Zero	No specific requirement.					
	_HID	SONY132A	The sensor hardware identifier, provided by Sensor Vendor. <table border="1"><tr><td>CMOS Sensor</td><td>Hardware ID</td></tr><tr><td>IMX214</td><td>SONY214A</td></tr><tr><td>IMX132</td><td>SONY132A</td></tr></table>	CMOS Sensor	Hardware ID	IMX214	SONY214A	IMX132
CMOS Sensor	Hardware ID							
IMX214	SONY214A							
IMX132	SONY132A							
_CID	Same as _HID	Same as _HID						
_UID	ONE	No specific requirement.						
_DDN	SONY IMX132	Device Name should be defined as sensor OEM company and sensor name. <table border="1"><tr><td>CMOS sensor</td><td>Device Name</td></tr><tr><td>IMX214</td><td>"SONY IMX214"</td></tr><tr><td>IMX132</td><td>"SONY IMX132"</td></tr></table>	CMOS sensor	Device Name	IMX214	"SONY IMX214"	IMX132	"SONY IMX132"
CMOS sensor	Device Name							
IMX214	"SONY IMX214"							
IMX132	"SONY IMX132"							
_STA	0xF							
Resource list (_CRS)	I2C for CMOS	I2cSerialBus(0x0036, ControllerInitiated, 0x00061A80, AddressingMode7Bit, "_SB.PCIO.I2C3", 0x00, ResourceConsumer,,)	The I2C used by COMS sensor. The I2C slave address depends on the sensor module. IMX214 Slave address: 0x1A IMX132 Slave address: 0x36					
	Reset Pin GPIO	GpioIo(Exclusive, PullDefault, 0x0000, 0x0000, IoRestrictionOutputOnly,	GPIO_67:CAM_1_RST_N					

Category	ACPI Items	Requirement	Comment
		<pre>"_SB.GPO0",0x00,ResourceConsumer,,) { // Pin list 0x0043 }</pre>	
	_PLD	<pre>{ /* 0000 */ 0x82, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0008 */ 0x61, 0x0C, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, /* 0010 */ 0xFF, 0xFF, 0xFF, 0xFF }</pre>	<p>The PLD is corresponding of the platform board itself including the rear or front information. In Intel RVP, it is defined as:</p> <p>Rear Sensor:</p> <pre>{ /* 0000 */ 0x82, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0008 */ 0x69, 0x0C, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, /* 0010 */ 0xFF, 0xFF, 0xFF, 0xFF }</pre> <p>Front Sensor:</p> <pre>{ /* 0000 */ 0x82, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, /* 0008 */ 0x61, 0x0C, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, /* 0010 */ 0xFF, 0xFF, 0xFF, 0xFF }</pre>

7.6.2 Device Specific Method (_DSM)

Camera sensor drivers get the camera functionality settings and Hardware configuration details by the sensor module interface and defined in the _DSM. The interfaces and parameters are defined as follow:

Basic Interface:

```
#define DSDT_CAM_HWID      { 377BA76A-F390-4AFF-AB38-9B1BF33A3015 }
#define DSDT_CAM_I2C         { 26257549-9271-4CA4-BB43-C4899D5A4881 }
#define DSDT_CAM_GPIO        { 79234640-9E10-4FEA-A5C1-B5AA8B19756F }
#define DSDT_CAM_ROM         { E770AB0F-2644-4BAB-8628-D62F1683FB9D }
#define DSDT_CAM_CMOS        { 3C62AAAA-D8E0-401A-84C3-FC05656FA28C }
#define DSDT_CAM_MODULE       { 822ACE8F-2814-4174-A56B-5F029FE079EE }
#define DSDT_CAM_CUSTOM       { 2959512A-028C-4646-B73D-4D1B5672FAD8 }
#define DSDT_CAM_MIPPORT      { EA3B7BD8-E09B-4239-AD6E-ED525F3F26AB }
#define DSDT_CAM_VCM          { 75C9A639-5C8A-4A00-9F48-A9C3B5DA789F }
#define DSDT_CAM_FUNC         { B65AC492-9E30-4D60-B5B2-F497C790D9CF }
#define DSDT_CAM_SNSR_CAL     { E9914EB6-592B-4228-BA5D-
  A0994BCB20DD }
```



```
#define DSDT_CAM_FLASH_FUN           { 5E95D048-B1DE-4590-947D-
CC699940CEE4 }

#define DSDT_CAM_INTF_VER            { 1EA54AB2-CD84-48CC-9DD4-
7F594EC3B015 }

#define DSDT_CAM_SNSR_MCLK           { 8DBE2651-70C1-4C6F-AC87-
A37CB46E4AF6 }
```

Table 7.1 Camera Module Interfaces and Settings

Resource	Details	Note	
DSDT_CAM_HWID	Hardware ID Each sensor module will have a Hardware ID for installer. For example, IMX214 hardware ID is ACPI\SONY214A. // For example – IMX214 in Intel FFRD If(LEqual(Arg0, ToUUID("377BA76A-F390-4AFF-AB38-9B1BF33A3015"))) //HWID { Return("SONY214A") }	Max Length: 16 Type: Char " SONY214A" CMOS sensor	Hardware ID
		IMX214	"SONY214A"
		IMX132	"SONY132A"
DSDT_CAM_I2C	1) I2C total number: the number of i2c that module has. For example imx214 module, it has three i2c resources: cmos (general) i2c, and vcm i2c. Each i2c will have an item data. 2) I2C resource data format: <u>(Bus)</u> I2C bus: for most cases, camera is using i2c bus 4 <u>(Func)</u> Function <u>(Addr)</u> Slave Address: The value is 1 bit right shift of write/read address. For example IMX214, the write/read address is 0x34/0x35, so the address value here should be 0x1A <u>(Speed)</u> Speed: Unused	Item Data Format: <u>bit</u> 31 24 23 16 15 8 7 0 0x Bus Speed Addr Func ↓ Function: #define I2C_GENERAL 0 #define I2C_VCM 1 #define I2C_EEPROM 2	

Resource	Details	Note																
	<pre>// For example – IMX214 in Intel RVP If(LEqual(Arg0, ToUUID("26257549-9271-4CA4-BB43-C4899D5A4881"))) //I2c function { if(LEqual(Arg2, 1)) // I2C total number { Return(0x01); } if(LEqual(Arg2, 2)) // General (CMOS) I2C { Return(0x02001A00); // I2C Bus 2, I2C Slave Address: 0x1A } }</pre>																	
DSDT_CAM_GPIO	<p>1) GPIO total number: the number of gpio that module using. For example IMX214 module, it has one gpio resources: RST pin. Each gpio pin resource will have an item data.</p> <p>2) GPIO resource data format</p> <ul style="list-style-type: none"> (<u>Func</u>) Function (<u>PinNo</u>) gpio pin no (<u>Active</u>) the pin value for camera working (<u>Initial</u>) Initial value for board power on 	<p>Item Data Format:</p> <table border="0"> <tr> <td>bit</td> <td>31</td> <td>24</td> <td>23</td> <td>16</td> <td>15</td> <td>8</td> <td>7</td> </tr> <tr> <td>0</td> <td> </td> </tr> </table> <p><u>0x</u> Active Initial PinNo Func Function</p> <pre>#define GPIO_RESET 0 #define GPIO_PWDN 1 #define GPIO_STROBE 2 #define GPIO_TORCH 3 #define GPIO_FLASH 4 #define GPIO_INDICATOR_REAR 5 #define GPIO_INDICATOR_FRONT 6</pre> <p>Pin Value:</p> <p>High 1, Low 0</p>	bit	31	24	23	16	15	8	7	0							
bit	31	24	23	16	15	8	7											
0																		
	<pre>// For example – IMX214 in Intel FFRD. Suppose the Reset pin connected with GPIO 24. If(LEqual(Arg0, ToUUID("79234640-9E10-4FEA-A5C1-B5AA8B19756F"))) //GPIO function { if(LEqual(Arg2, 1)) // GPIO total number { Return(0x01); } if(LEqual(Arg2, 2)) // GPIO 0 - RST { Return(0x01004100); // 1 - (active)normal } }</pre>																	



Resource	Details	Note				
DSDT_CAM_MIPIORT	<p>Sensor MIPI Port Configuration</p> <p>We can configure that which mipi port the sensor connected and how many lanes used.</p>	Item data format: Bit[15:12]: MIPI define place mipi defined in driver; mipi defined in dsm Bit[7:4]: MIPI lanes 1-1lanes; 2-2lanes; 4-4lanes Bit[3:0]: MIPI ports 0-port0; 1-port1; 2-port2				
	<p>// For example – IMX214 in Intel FFRD</p> <pre>If(LEqual(Arg0, ToUUID("EA3B7BD8-E09B-4239-AD6E-ED525F3F26AB "))) //MIPI Port { Return(0x1040) }</pre>					
DSDT_CAM_ROM	<p>Sensor Module ROM</p> <p>There may have two kinds of ROM for saving module calibration data OTP and EEPROM. OTP is with CMOS, and EEPROM is with module. But there are some sensor modules without module roms. For example, IMX132 in Intel RVP does not have any available Rom.</p>	#define ROM_NONE 0 #define ROM OTP 1 #define ROM EEPROM 2 #define ROM OTP ACPI ACPI 3 #define ROM ACPI 4				
	<p>// For example – IMX132 in Intel FFRD</p> <pre>If(LEqual(Arg0, ToUUID("E770AB0F-2644-4BAB-8628-D62F1683FB9D"))) //ROM { Return(0x00); // ROM_NONE }</pre>					
DSDT_CAM_CMOS	<p>Sensor CMOS name</p> <p>The different CMOS sensor with same setting may use same driver. So in DSDT this information will be changed based on different sensors.</p>	Max Length: 64 Type: Char <table border="1"><thead><tr><th>Platform</th><th>Module Name</th></tr></thead><tbody><tr><td>Intel RVP</td><td>"IMX214"</td></tr></tbody></table>	Platform	Module Name	Intel RVP	"IMX214"
Platform	Module Name					
Intel RVP	"IMX214"					
	<p>// For example – IMX214 in Intel FFRD</p> <pre>If(LEqual(Arg0, ToUUID("3C62AAAA-D8E0-401A-84C3-FC05656FA28C "))) //Sensor CMOS Name { Return("IMX214") }</pre>					

Resource	Details	Note
DSDT_CAM_MODULE	<p>Sensor Module name The same CMOS sensor will be in different camera module. The different camera module may have different tuning data. So the module name should be declared in DSDT table for driver to query and so that driver can load the corresponding tuning data.</p>	Max Length: 64 Type: Char
	<pre>// For example – IMX214 in Intel FFRD If(LEqual(Arg0, ToUUID("822ACE8F-2814-4174-A56B-5F029FE079EE"))) //Sensor Module Name { Return("P13N05BA") }</pre>	
DSDT_CAM_CUSTOMER	<p>Customer/Platform Info The same module in different customer or platform will also have different software configuration. So that the platform information also need to be declared in DSDT.</p>	Max Length: 64 Type: Char The platform name is defined as: "INTEL_RVP" "ASUS" "ACER" "LENOVO" Different customer should define the company name in this field.
	<pre>// For example – IMX214 in Intel RVP If(LEqual(Arg0, ToUUID("2959512A-028C-4646-B73D-4D1B5672FAD8"))) //Customer/platform { Return("INTEL_RVP") }</pre>	
DSDT_CAM_VCM	<p>VCM module name The same cmos sensor in different module may use different VCM motor.</p>	
	<p>// For example – IMX214 in Intel RVP without VCM.</p>	
	<pre>If(LEqual(Arg0, ToUUID("75C9A639-5C8A-4A00-9F48-A9C3B5DA789F ")) { Return("DW9714") }</pre>	



Resource	Details	Note
DSDT_CAM_FUNC	Sensor HW Function Hardware orientation 0 or 180 degree Flash enable/disable	Bit 1:0 – Orientation 00 – #define DEGREE_0 01 – #define DEGREE_180 10 – Reserved 11 -- Reserved Bit 3:2 – Flash LED Enabled: 00 – #define FLASH_DEFAULT (Driver defined) 01 – Reserved 10 – #define FLASH_DISABLE (Without Flash) 11 – #define FLASH_ENABLE (With Flash) Bit 31:4 -- Reserved
	// For example – IMX214 in Intel RVP <pre>If(LEqual(Arg0, ToUUID("B65AC492-9E30-4D60-B5B2-F497C790D9CF ")) { Return(0x00000000) }</pre> <p>It means: HW 0 degree orientation, and FLASH_DEFAULT (Driver defined)</p>	
DSDT_CAM_SNSR_CAL	Sensor Calibration Data It will be valid, when DSDT_CAM_ROM is ROM_ACPI.	The buffer length: 545 #define CALIBRATION_LENGTH 545
DSDT_CAM_FLASH_FUN	Flash HW Function 1. Hardware Heat Protection. 2. Flash peak current limit.	Bit 0 – Heat Protection Enable/Disable 0 – Disable 1 – Enable Bit 3:1 – Flash Peak Current Limit: For LM3643 000 – #define PEAK_LIMIT_1A9 0 001 – #define PEAK_LIMIT_2A8 1 (Default value) Other value is reserved. Bit 31:4 -- Reserved
	// For example – LM3643 in Intel RVP <pre>If(LEqual(Arg0, ToUUID("B65AC492-9E30-4D60-B5B2-F497C790D9CF ")) { Return(0x00000001) }</pre> <p>It means: Heat protection is enabled, the current limit is 2.8A.</p>	
DSDT_CAM_INTF_VER	Camera DSDT interface version.	Return value: 0, 1, or 2.

Resource	Details	Note
DSDT_CAM_SNSR_MCLK	Camera Sensor MCLK Index The index of input clock resource.	Return value: 0 – CLK0 1 – CLK1

7.6.3 Control Logic ASL Sample Code

```

Device(PMC0)
{
    Name(_ADR, Zero)
    Name(_HID, "INT3472")
    Name(_CID, "INT3472")
    Name(_DDN, "INCL-CRDD")
    Name(_UID, "1")
    Method(_CRS, 0x0, NotSerialized)
    {
        Name(SBUF, ResourceTemplate()
        {
            GpioIo(Exclusive, PullDefault, 0x0000, 0x0000,
IoRestrictionOutputOnly,
                    "\_SB.GPO0", 0x00, ResourceConsumer, ,
            {
                // Pin list
                0x35      // GPIO_65
            }
            GpioIo(Exclusive, PullDefault, 0x0000, 0x0000,
IoRestrictionOutputOnly,
                    "\_SB.GPO0", 0x00, ResourceConsumer, ,
            {
                // Pin list
                0x3B      // GPIO_71
            }
        })
        Return (SBUF)
    }
    Method(_STA, 0, NotSerialized) // _STA: Status
    {
        Return (0x0F)
    }
    Method(CLDB)
    {
        Name(PAR, Buffer(0x20))
        {
            0x00,      //Version
            0x01,      //Control logic Type 0: UNKNOWN 1:
DISCRETE 2: PMIC TPS68470 3: PMIC uP6641
            0x01,      //Control logic ID: Control Logic 1
        }
    }
}

```



```
0x70,           //CRD board Type, 0: UNKNOWN  0x20: CRD-D
0x30: CRD-G  0x40: PPV 0x50: CRD-G2  0x70: CRD_G_BXT
               0x00, 0x00, 0x00, 0x00,           // Reserved
               0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
// Reserved
               0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
// Reserved
               0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
// Reserved
}
      Return (PAR)
}
Method (_DSM, 4, NotSerialized) // _DSM: Device-Specific
Method
{
    If (LEqual (Arg0, ToUUID("79234640-9E10-4FEA-A5C1-
B5AA8B19756F")))
    {
        If (LEqual (Arg2, One))
        {
            Return (0x02)           // number
        }
        If (LEqual (Arg2, 0x02))
        {
            Return (0x01004100)    // RESET
        }
        If (LEqual (Arg2, 0x03))
        {
            Return (0x01004701)    // POWER DOWN
        }
    }
    Return (Zero)
}
}
```

7.6.4 Camera Module ASL Sample Code

```
Device (CAM0)
{
    Name (_ADR, Zero) // _ADR: Address
    Name (_HID, "INT3471") // _HID: Hardware ID
    Name (_CID, "INT3471") // _CID: Compatible ID
    Name (_SUB, "INTL0000") // _SUB: Subsystem ID
    Name (_DDN, "SONY IMX135") // _DDN: DOS Device Name
```

```
Name (_UID, One) // _UID: Unique ID

/* need to be update after power on
Name (_DEP, Package ()) // _DEP: Dependencies
{
    //PMIC device
}

Name (_PR0, Package (0x03) // _PR0: Power Resources for D0
{
    // Power and Clock
}
*/
Name (PLDB, Package (0x01)
{
    Buffer (0x14)
    {
        /* 0000 */ 0x82, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
        /* 0008 */ 0x69, 0x0C, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00,
0x00,
        /* 0010 */ 0xFF, 0xFF, 0xFF, 0xFF
    }
})
Method (_PLD, 0, Serialized) // _PLD: Physical Location of
Device
{
    Return (PLDB)
}

Method (_STA, 0, NotSerialized) // _STA: Status
{
    Return (0x0F)
}

/* need to be update after power on
Method (_PS3, 0, Serialized) // _PS3: Power State 3
{
}
Method (_PS0, 0, Serialized) // _PS0: Power State 0
{
}
*/
}
```



```
Method (_CRS, 0, NotSerialized) // _CRS: Current Resource
Settings
{
    Name (SBUF, ResourceTemplate ())
    {
        GpioIo (Exclusive, PullDefault, 0x0000, 0x0000,
IoRestrictionOutputOnly,
                "\_SB.GPO0", 0x00, ResourceConsumer, ,
                )
        {
            0x0037 // Reset Pin
GPIO_67:GP_CAMERASB05:C_RESET_N<3>
        }
        GpioIo (Exclusive, PullDefault, 0x0000, 0x0000,
IoRestrictionOutputOnly,
                "\_SB.GPO0", 0x00, ResourceConsumer, ,
                )
        {
            0x003C // PowerOn Pin
GPIO_72:GP_CAMERASB10:C_PWRON<3>
        }
        //I2C for CMOS
        I2cSerialBus (0x0010, ControllerInitiated, 0x00061A80,
                        AddressingMode7Bit, "\_SB.PCI0.I2C2",
                        0x00, ResourceConsumer, ,
                        )
        I2cSerialBus (0x000E, ControllerInitiated, 0x00061A80,
                        AddressingMode7Bit, "\_SB.PCI0.I2C2",
                        0x00, ResourceConsumer, ,
                        )
    }
    Return (SBUF)
}

Method (_DSM, 4, NotSerialized) // _DSM: Device-Specific Method
{
    If (LEqual (Arg0, Buffer (0x10)
    {
        /* 0000 */ 0x6A, 0xA7, 0x7B, 0x37, 0x90, 0xF3,
0xFF, 0x4A,
        /* 0008 */ 0xAB, 0x38, 0x9B, 0x1B, 0xF3, 0x3A,
0x30, 0x15
    }))
    {
        Return ("INT3471") //DSDT_CAM_HWID
    }
}
```

```
        If (LEqual (Arg0, Buffer (0x10)
                    {
                        /* 0000 */ 0xAA, 0xAA, 0x62, 0x3C, 0xE0, 0xD8,
0xA1A, 0x40,
                        /* 0008 */ 0x84, 0xC3, 0xFC, 0x05, 0x65, 0x6F,
0xA2, 0x8C
                    } ))
    {
        Return ("IMX135") //DSDT_CAM_CMOS
    }

    If (LEqual (Arg0, Buffer (0x10)
                {
                    /* 0000 */ 0x8F, 0xCE, 0x2A, 0x82, 0x14, 0x28,
0x74, 0x41,
                    /* 0008 */ 0xA5, 0x6B, 0x5F, 0x02, 0x9F, 0xE0,
0x79, 0xEE
                } ))
    {
        Return ("13P2BAD33") //DSDT_CAM_MODULE
    }

    If (LEqual (Arg0, Buffer (0x10)
                {
                    /* 0000 */ 0x2A, 0x51, 0x59, 0x29, 0x8C, 0x02,
0x46, 0x46,
                    /* 0008 */ 0xB7, 0x3D, 0x4D, 0x1B, 0x56, 0x72,
0xFA, 0xD8
                } ))
    {
        Return ("BXT") //DSDT_CAM_CUSTOM
    }

    If (LEqual (Arg0, Buffer (0x10)
                {
                    /* 0000 */ 0xD8, 0x7B, 0x3B, 0xEA, 0x9B, 0xE0,
0x39, 0x42,
                    /* 0008 */ 0xAD, 0x6E, 0xED, 0x52, 0x5F, 0x3F,
0x26, 0xAB
                } ))
    {
        Return (0x1046) //DSDT_CAM_MIPIPORT
    }

    If (LEqual (Arg0, Buffer (0x10)
    {
```



```
        /* 0000 */ 0x92, 0xC4, 0x5A, 0xB6, 0x30, 0x9E,
0x60, 0x4D,
        /* 0008 */ 0xB5, 0xB2, 0xF4, 0x97, 0xC7, 0x90,
0xD9, 0xCF
    } ))
{
    Return (0xC)           //DSDT_CAM_FUNC: Flash Enabled
}

If (LEqual (Arg0, Buffer (0x10)
{
    /* 0000 */ 0x0F, 0xAB, 0x70, 0xE7, 0x44, 0x26,
0xAB, 0x4B,
    /* 0008 */ 0x86, 0x28, 0xD6, 0x2F, 0x16, 0x83,
0xFB, 0x9D
} ))
{
    Return (Zero)          //DSDT_CAM_ROM
}

If (LEqual (Arg0, Buffer (0x10)
{
    /* 0000 */ 0xB2, 0x4A, 0xA5, 0x1E, 0x84, 0xCD,
0xCC, 0x48,
    /* 0008 */ 0x9D, 0xD4, 0x7F, 0x59, 0x4E, 0xC3,
0xB0, 0x15
} ))
{
    Return (One)           //DSDT_CAM_INTF_VER
}

If (LEqual (Arg0, Buffer (0x10)
{
    /* 0000 */ 0x39, 0xA6, 0xC9, 0x75, 0x8A, 0x5C,
0x00, 0x4A,
    /* 0008 */ 0x9F, 0x48, 0xA9, 0xC3, 0xB5, 0xDA,
0x78, 0x9F
} ))
{
    Return ("AD5816")      //DSDT_CAM_VCM
}

If (LEqual (Arg0, Buffer (0x10)
{
    /* 0000 */ 0x51, 0x26, 0xBE, 0x8D, 0xC1, 0x70,
0x6F, 0x4C,
```

```

        /* 0008 */ 0xAC, 0x87, 0xA3, 0x7C, 0xB4, 0x6E,
0x4A, 0xF6
    } ))
{
    Return (0x02)      // MCLK2 --> OSC_CLKOUT_R_2
}

If (LEqual (Arg0, Buffer (0x10)
{
    /* 0000 */ 0x49, 0x75, 0x25, 0x26, 0x71, 0x92,
0xA4, 0x4C,
    /* 0008 */ 0xBB, 0x43, 0xC4, 0x89, 0x9D, 0x5A,
0x48, 0x81
} ))
{
    If (LEqual (Arg2, One)) //DSDT_CAM_I2C
    {
        Return (0x02)
    }

    If (LEqual (Arg2, 0x02))
    {
        Return (0x03001000)
    }
    If (LEqual (Arg2, 0x03))
    {
        Return (0x02000E01)
    }
}

If (LEqual (Arg0, Buffer (0x10)
{
    /* 0000 */ 0x40, 0x46, 0x23, 0x79, 0x10, 0x9E,
0xEA, 0x4F,
    /* 0008 */ 0xA5, 0xC1, 0xB5, 0xAA, 0x8B, 0x19,
0x75, 0x6F
} ))
{
    If (LEqual (Arg2, One)) //DSDT_CAM_GPIO
    {
        Return (0x02)
    }

    If (LEqual (Arg2, 0x02))
    {
        Return (0x01004300)
    }
}

```



```
        If (LEqual (Arg2, 0x03))
        {
            Return (0x01004801)
        }
    }

    Return (Zero)
}
}
```

7.6.5 Flash ASL Sample Code

```
//Flash Device TPS61311
Device (STRB)
{
    Name (_ADR, Zero) // _ADR: Address
    Name (_HID, "INT3481") // _HID: Hardware ID
    Name (_CID, "INT3481") // _CID: Compatible ID
    Name (_SUB, "INTL0000") // _SUB: Subsystem ID
    Name (_DDN, "Flash TPS61311") // _DDN: DOS Device Name
    Name (_UID, 0) // _UID: Unique ID
    Name (PLDB, Package (0x01)
    {
        Buffer (0x14)
        {
            /* 0000 */ 0x82, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
/* ..... */
            /* 0008 */ 0x69, 0x0C, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00,
/* i..... */
            /* 0010 */ 0xFF, 0xFF, 0xFF, 0xFF
/* .... */
        }
    })
    Method (_PLD, 0, Serialized) // _PLD: Physical Location of Device
    {
        Return (PLDB) /* \_SB_.PCI0.I2C2.FLASH.PLDB */
    }
    Method (_STA, 0, NotSerialized) // _STA: Status
    {
        Return (0x0F)
    }
    Method (_CRS, 0, NotSerialized) // _CRS: Current Resource Settings
    {
```

```

Name (SBUF, ResourceTemplate ())
{
    GpioIo (Exclusive, PullDefault, 0x0000, 0x0000,
IoRestrictionOutputOnly,
        "\_SB.GPO0", 0x00, ResourceConsumer, ,
        )
        {
            // Pin list
            0x36      ///Stobe Pin: GP_CAMERASB04
        }
    I2cSerialBus (0x0033, ControllerInitiated, 0x00061A80,
        AddressingMode7Bit, "\_SB.PCI0.I2C2",
        0x00, ResourceConsumer, ,
        )
    }
}
Return (SBUF)
}
Method (_DSM, 4, NotSerialized) // _DSM: Device-Specific Method
{
    If (LEqual(Arg0, ToUUID ("377ba76a-f390-4aff-ab38-
9b1bf33a3015")))
    {
        Return ("INT3481") //Hardware ID
    }
    If (LEqual(Arg0, ToUUID ("3c62aaaa-d8e0-401a-84c3-
fc05656fa28c")))
    {
        Return ("TPS61311") // DSDT_FLASH_CMOS
    }
    If (LEqual(Arg0, ToUUID ("2959512a-028c-4646-b73d-
4d1b5672fad8")))
    {
        Return ("BXT") //Platform Info
    }
    If (LEqual(Arg0, ToUUID ("26257549-9271-4ca4-bb43-
c4899d5a4881")))
    {
        If (LEqual (Arg2, One)) //DSDT_FLASH_I2C
        {
            Return (One)
        }
        If (LEqual (Arg2, 0x02))
        {
            Return (0x02003300)
        }
    }
}

```



```
If (LEqual(Arg0, ToUUID ("79234640-9e10-4fea-a5c1-
b5aa8b19756f")))
{
    If (LEqual (Arg2, One)) //DSDT_FLASH_GPIO
    {
        Return (0x01)
    }
    If (LEqual (Arg2, 0x2))
    {
        Return (0x01003604)
    }
    Return (Zero)
}
}
```

§

**8**

Security Considerations

This chapter is intended to help IA FW developers by providing recommendations and requirements for making IA FW based on SoC.

8.1 SoC Security Mechanism for SSA Memory Protection

IA FW is friendly for SoC and expected to be the only agent that configures many aspects of the system. For example, IA FW is expected to initialize the memory controller correctly without aliasing. In general, these settings are expected to remain static once set. To increase system robustness, many of these settings were previously protected from further change with “Lock” bits or Write-Once bits. This disadvantage of locks bits is that they prevent later changes by trusted agents such as microcode (μcode).

SoC general policy is what was previously an IA FW set lock bit now has IA Boot access but not untrusted IA access.

SoC provide HW security mechanism which is designed to provide access control for data stored in DRAM. Security checks happen in the Physical Address Space, but only for request on the way to DRAM rather than devices/MMIO.

In addition, for reliable operation and security, it is strongly recommended that IA FW should set the following bits:

- Lock GGC from writes by setting the B0.D2.F0 register offset 050 [0] = ‘1b’.
- Lock PAVPC by setting the B0.D2.F0 register offset 058 [0] = ‘1b’.
- Lock BDSM by setting the B0.D2.F0 register offset 0B0h [0] = ‘1b’.
- Lock BGSM by setting the B0.D2.F0 register offset 0B4h [0] = ‘1b’.

8.2 Isolated Memory Regions (IMRs)

Isolated memory regions (IMRs), which are areas carved out of DRAM that are accessible only to certain Bunit agents or threads. The number of regions that the Bunit supports is a build time parameter and up to 20 IMR regions can be supported.

Each region monitors accesses within its pre-programmed memory range and either permits or denies read and write accesses from different system agents. If a request is denied access to a region due to violating the IMR policy rules, it is not allowed access to system memory, the Agent ID and Thread ID are stored in the Bunit, and the P-Unit is notified that a violation has occurred.

8.2.1 IA FW Responsibilities

IA FW would send the IMR base to the TXE and returned with size. IA FW would then mark the region as stolen memory inside system memory map.

§



9 Boot Guard 2.0

The primary goal of Boot Guard 2.0 is to converge on secure boot flows between Atom and Big core with scalable implementation across multiple FW storage media.

9.1 Architecture

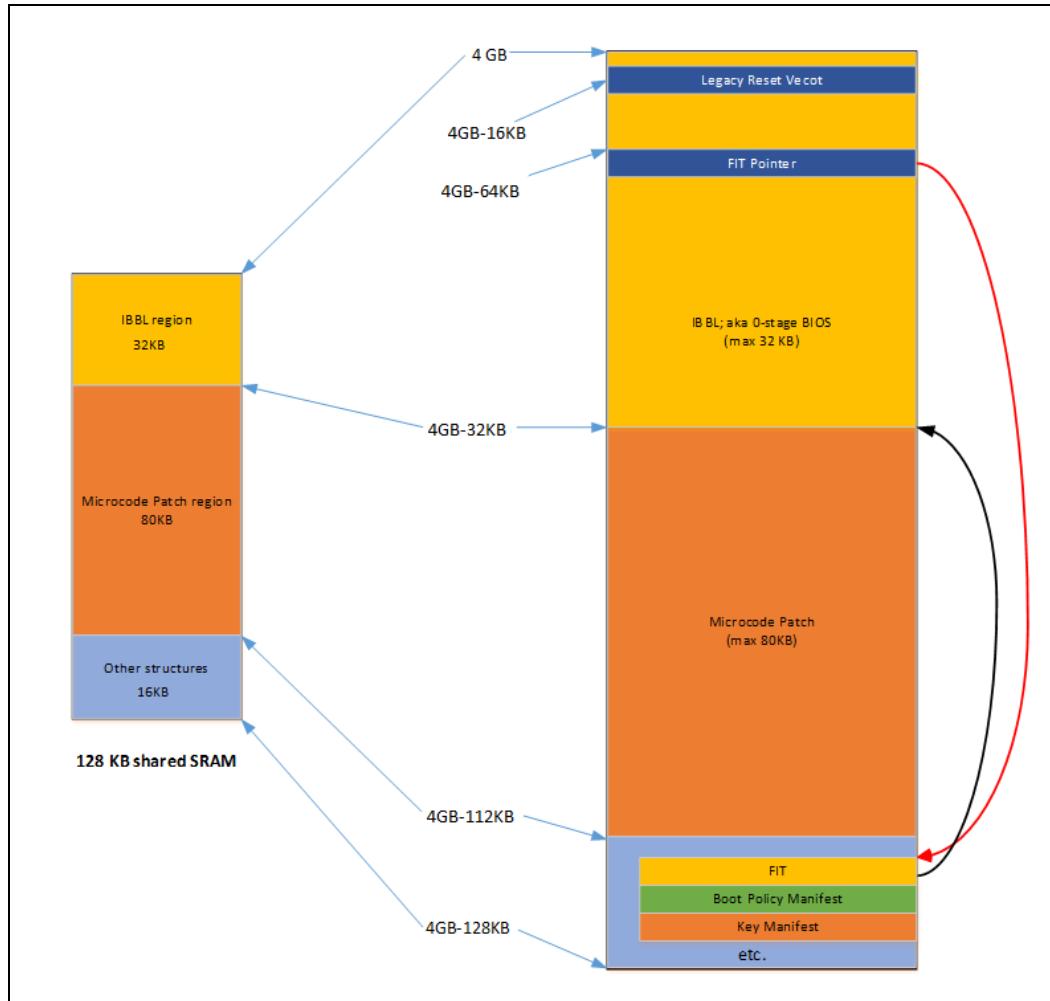
In Boot Guard 2.0, secure boot is rooted from TXE using ROM based RSA key. Shown below is the high level FW flow with Boot Guard 2.0.

1. TXE selects the FW boot media (eMMC/SPI) based on HW strapping.
2. TXE sets up 128KB of TXE SRAM as shared SRAM with read-only access to host CPU at below 4GB address space.
3. TXE selects and fetches the right microcode patch and authenticated IBBL from FW boot media into the shared SRAM.
4. TXE constructs FIT in shared SRAM and powers up the north cluster.
5. Host CPU consumes microcode patch using FIT, and passes control to the IBBL.
6. IBBL starts executing from shared SRAM. IBBL sets up CPU cache as RAM (CAR) and copies itself to CAR and resumes execution from CAR. IBBL also copies necessary FIT table entries to CAR area as the FIT table in shared SRAM will be overwritten by the next step.
7. IBBL requests TXE to load IBB from FW boot media to shared SRAM.
8. TXE fetches and authenticates the IBB from FW Boot Media and copies it to shared SRAM, in chunks. The check size will be 128KB/64KB/32KB. The IBBL and TXE use ring buffer protocol so that the whole IBB is transferred from FW Boot Media, to shared SRAM, to CAR area.
9. After the entire IBB is copied to CAR, the IBBL sends a message to TXE indicating the end of IBB copy process and passes control to IBB.
10. When TXE receives the message that IBBL has picked up the last chunk of IBB, it fetches the SMIP for IBB and puts into shared SRAM. IBB can derive the location of SMIP in shared SRAM through FIT table.
11. IBB configures the RAM and starts executing from RAM.
12. TXE closes the shared SRAM from host CPU access.
13. IBB handshakes with TXE to take ownership of FW Boot Media.
14. IBB fetches and authenticates the OBB from FW Boot Media into DRAM.
15. IBB give control to OBB and the boot process continuous to OS boot loader.

9.1.1 Ring Buffer Protocol

Shown below is the shared SRAM layout at host CPU reset, prepared by TXE. Since the shared SRAM is mapped below 4GB, host CPU comes out of reset and continues with conventional boot process by loading microcode using FIT and executing IBBL from shared SRAM. But the shared SRAM size is 128KB and cannot accommodate whole IBB. Ring Buffer Protocol (RBP) has been defined for hand shake between TXE and IBBL, so that IBB can be copied in 128KB chunks from FW Boot Media to shared SRAM to CAR area.

Figure 8-1. Shared SRAM Layout at CPU Reset



CSE_TO_HOST and HOST_TO_CSE are the two 32-bit registers defined in HECI PCI config space for RBP. Shown below is the sequence of steps involved in RBP.

1. IBBL figures out the amount of cache available for RBP, and determines the value of MAX IBB SIZE ALLOWED (1KB granularity), and fills the MISA field in HOST_TO_CSE register.
2. IBBL fills the UCODE STATUS field in HOST_TO_CSE register; 1=patch successful; 0=patch failed.
3. IBBL indicates to TXE by setting IBBL READY FOR RBP in HOST_TO_CSE register that it is ready to load IBB via the RBP.
4. IBBL polls on ACTUAL IBB SIZE field to become non-zero in CSE_TO_HOST register.
5. TXE gets the actual IBB size from eMMC and populates AIS field in CSE_TO_HOST.
6. TXE populates NOC field in CSE_TO_HOST. This value is hard coded in TXE FW.



7. Assuming NOC = 1 (128KB chunks), the hand shake between TXE and IA FW for copying IBB to CAR works as follows. The same operation works with more chunks at a time using the rest of CHx, HCx bits, when NOC is 2/3.
 - If CHO and HCO are equal (this is the starting condition), TXE will read the chunk from eMMC and copies to shared SRAM and flips CHO.
 - If CHO and HCO are not equal (means TXE has just copied new chunk in shared SRAM), IBBL will copy the chunk from SRAM to CAR and flips HCO.
 - The process continues until all the chunks of IBB are read from eMMC and copied over to CAR. Note that, IBBL should only read the reaming IBB size in 1KB granularity for the last chunk, instead of coping the whole chunk.

9.1.2 FIT (Firmware Interface Table)

Goldmont CPU looks for a FIT when it comes out of reset. The FIT pointer is always located at 4G-64 location in the CPU address map (0xFFFFFFF0). The FIT is a variable length structure, and it could contain several types of records. TXE creates FIT in the shared SRAM, and put its pointer at 0xFFFFFFF0 before CPU reset is de-asserted. Shown below is the list of FIT entry types defined for Intel platforms. The FIT entries created by TXE for Apollo Lake are highlighted with an underline.

Table 8-1. FIT Entry Types

FIT Entry Type	Description
0x00	FIT Header Entry
0x01	Intel Reserved (used for reset ucode patch)
0x02	Startup AC Module Entry
0x07	BIOS Startup Module Entry
0x08	TPM Policy Record
0x09	BIOS Policy Record
0x0a	TXT policy Record
<u>0x0b</u>	<u>Key Manifest Record</u>
0x0c	Boot Policy Manifest
<u>0x10</u>	<u>TXE Secure Boot</u>
0x11 – 0x2E	Intel Reserved.
0x2f	JMP \$ debug policy
0x30 – 0x70	Reserved for Platform Manufacturer use
0x71 – 0x7E	Intel Reserved
0x7F	Unused Entry, Skip

There will be multiple TXE Secure Boot entries (type = 0x10) in FIT. The reserved filed in FIT entry (bit[31:24]) is used to identify the sub-type of the TXE Secure Boot entries as follows.



FIT Entry Format:

Qword	63:56	55	54:48	47:32	31:24	23:0
N	Address					
N+1	Checksum	C_V	Type	Version	Reserved	Size

[N+1][54:48] = 0x10 (i.e type = 0x10 = TXE Secure Boot)

[N+1][31:24] = FIT Entry Sub-type

0 = unused/reserved

1 = Key Hash 1

2 = TXE Measurement hash

3 = Boot Policy

4 = Other Boot Policy

5 = OEM SMIP

6 = MRC Training Data

7 = IBBL Hash

8 = IBB Hash

9 = OEM ID

10 = OEM SKU ID

11 onwards = reserved

9.1.3 Measured Boot

MB bit in OEM IFP material (present in FIT) indicates if Measured Boot is enabled. If enabled, IBB needs to handle the following tasks.

1. Initialize TPM
2. Obtain the hash of TXE FW from FIT
3. Obtain the hash of IBBL from FIT
4. Extend the measurement of TXE FW and IBBL into TPM.
5. After RBP, extend the measurement of IBB into TPM.
6. After loading OBB to RAM, extend the measurement of OBB into TPM.



9.1.4 IA FW Requirements

1. Generate IBBL of 32KB size to handle below functionalities.
 - Reset vector code.
 - Initialize CAR, self-copy to CAR and start executing from CAR.
 - Copy IBB using RBP from shared SRAM to CAR and give control to IBB.
2. Add support in IBB to read OBB from FW storage media, authenticate OBB using TXE and give control to OBB.
3. Add support for S3 resume in IBBL and IBB flow.

9.1.5 BPM Metadata Structure

Structure of the Boot Policy Manifest Metadata.

Table 8-2. BPM Metadata Structure

Name	Offset	Size (bytes)	Description	BPM for APL
Extension Type	0	4	= 19 for Boot Policy metadata file extension	Mandatory
Extension Length	4	4	In bytes	Mandatory
NEM Data	8	4	Size of data region need by IBB In 4 K pages. E.g., value of 1 = 4096 bytes; 2 = 8092 bytes, and so on.	Optional
IBBL Hash Algo	12	4	IBBL hash algorithm	Mandatory
IBBL Hash Size	16	4	IBBL hash size	Mandatory
IBBL Hash	20	32	IBBL Hash	Mandatory
IBB Hash Algo	52	4	IBB hash algorithm	Mandatory
IBB Hash Size	56	4	IBB hash size	Mandatory
IBB Hash	60	32	IBB Hash	Mandatory
OBB Hash Algo	92	4	OBB hash algorithm	Mandatory
OBB Hash Size	96	4	OBB hash size	Mandatory
OBB Hash	100	32	OBB Hash	Mandatory
IBB Flags	132	4	Various flags directing the configuration and operation of this IBB.	Optional
IBB MCHBAR	136	8	MCHBAR address	Optional
IBB VT-d BAR	144	8	VTDPVCBAR	Optional
PMR-L Base	152	4	32-bit base of PMR-L	Optional



Name	Offset	Size (bytes)	Description	BPM for APL
PMR-L Limit	156	4	Limit of PMR-L. Register pointing to last address (limit) of the DMA-protected low memory region	Optional
PMR-H Base	160	8	64-bit base of PMR-H (HSW-ULT: Reserved – fail if non-zero)	Optional
PMR-H Limit	168	8	Limit of PMR-H. Register pointing to last address (limit) of the DMA-protected low memory region. (HSW-ULT: Reserved – fail if non-zero)	Optional
IBB Entry Point	176	4	Entry point for this IBB	Optional
IBB Segment Count (N)	180	4	Minimum N = 0 (APL could have N=0)	Optional
			Maximum N = 8	Optional
IBB Segment 0 Flags	184	4	Flags indicating how this segment is treated	Optional
IBB Segment 0 Base	188	4	Base of segment	Optional
IBB Segment 0 Size	192	4	Size of segment	Optional
...				Optional
BIOS Vendor Specified Attributes Size (N)		4	Size of Vendor Specific Attributes in bytes	Optional (Specific to IBV only)- Default is 0
			Max = 512 bytes	
BIOS Vendor Specified Attributes		N	Vendor specific attributes	Optional

§



10 Interrupts and IRQs

10.1 PCI IRQ Routing on Apollo Lake SoC

APL does not have wires for interrupt into APIC, but sideband messages. PCI Spec does not define how INTA-D map to IRQs in the system, its arbitrary and platform proprietary. OS does not care how INTA-D map to IRQs or what IRQs they map to or how they are delivered to the A/PIC. Windows* expects all INTA or all INTB for a given device map to the same IRQ.

APL has a mix of interrupt delivery methods:

- MSI
- Assert_IRQn
- Assert_INTA-D

10.1.1 MSI

MSI is naturally the preferred method for interrupt delivery if possible. MSI are enabled through the standard PCI config register (CAP_ID = 0x5)

Supported device in APL: NPK, Display, Iunit, Audio, TXE, xHCI

10.1.2 assert_IRQn

Devices configured as PCI or ACPI may support this IOSF sideband message.

IA FW should program the device for assigned IRQ number.

- For LPSS and SCC, IA FW should program PCICFGCTRL register.

Supported device in APL: DPTF, PMC, Audio, ISH, LPSS, SCC.

10.1.3 assert_INTA-D

Least preferred method as interrupts are limited to 8 IRQs, potential for sharing.

APL IOAPIC has 13 PIRx registers for mapping PIRQx. IA FW should program the corresponding PIRx register to map INTx to PIRQx (ITSS CR offset 3140h – 3158h)



Bit	Type	Reset	Description																				
15	RO	0	Reserved																				
14:12	RW	3h	<p>Interrupt D Pin Route (IDR): Indicates which PIRQ in PCH is connected to the INTD# pin reported for device X functions.</p> <table border="1"> <thead> <tr> <th>Bits</th><th>Pin</th><th>Bits</th><th>Pin</th></tr> </thead> <tbody> <tr> <td>0h</td><td>PIRQA#</td><td>4h</td><td>PIRQE#</td></tr> <tr> <td>1h</td><td>PIRQB#</td><td>5h</td><td>PIRQF#</td></tr> <tr> <td>2h</td><td>PIRQC#</td><td>6h</td><td>PIRQG#</td></tr> <tr> <td>3h</td><td>PIRQD#</td><td>7h</td><td>PIRQH#</td></tr> </tbody> </table> <p>Note: Refer to PCI Interrupt Route to PCI Device Number mapping table for the device number associated to IDR.</p>	Bits	Pin	Bits	Pin	0h	PIRQA#	4h	PIRQE#	1h	PIRQB#	5h	PIRQF#	2h	PIRQC#	6h	PIRQG#	3h	PIRQD#	7h	PIRQH#
Bits	Pin	Bits	Pin																				
0h	PIRQA#	4h	PIRQE#																				
1h	PIRQB#	5h	PIRQF#																				
2h	PIRQC#	6h	PIRQG#																				
3h	PIRQD#	7h	PIRQH#																				
11	RO	0	Reserved																				
10:8	RW	2h	Interrupt C Pin Route (ICR): See the IDR description. This field applies to INTC#.																				
7	RO	0	Reserved																				
6:4	RW	1h	Interrupt B Pin Route (IBR): See the IDR description. This field applies to INTB#.																				
3	RO	0	Reserved																				
2:0	RW	0h	Interrupt A Pin Route (IAR): See the IDR description. This field applies to INTA#.																				

For 8259 PIC, IA FW should program P[A-H] RC register (ITSS CR offset 3100h – 3107h) to map PIRQx to 8259 IRQs.

Bit	Type	Reset	Description																																				
7	RW	1	Interrupt Routing Enable (REN): When cleared, the corresponding PIRQ is routed to one of the legacy interrupts specified in bits[3:0]. When set, the PIRQ is not routed to the 8259.																																				
6:4	RO	0	Reserved																																				
IRQ Routing (IR):																																							
3:0	RW	0	<table border="1"> <thead> <tr> <th>Bits</th><th>Mapping</th><th>Bits</th><th>Mapping</th></tr> </thead> <tbody> <tr> <td>0000</td><td>Reserved</td><td>1000</td><td>Reserved</td></tr> <tr> <td>0001</td><td>Reserved</td><td>1001</td><td>IRQ9</td></tr> <tr> <td>0010</td><td>Reserved</td><td>1010</td><td>IRQ10</td></tr> <tr> <td>0011</td><td>IRQ3</td><td>1011</td><td>IRQ11</td></tr> <tr> <td>0100</td><td>IRQ4</td><td>1100</td><td>IRQ12</td></tr> <tr> <td>0101</td><td>IRQ5</td><td>1101</td><td>Reserved</td></tr> <tr> <td>0110</td><td>IRQ6</td><td>1110</td><td>IRQ14</td></tr> <tr> <td>0111</td><td>IRQ7</td><td>1111</td><td>IRQ15</td></tr> </tbody> </table>	Bits	Mapping	Bits	Mapping	0000	Reserved	1000	Reserved	0001	Reserved	1001	IRQ9	0010	Reserved	1010	IRQ10	0011	IRQ3	1011	IRQ11	0100	IRQ4	1100	IRQ12	0101	IRQ5	1101	Reserved	0110	IRQ6	1110	IRQ14	0111	IRQ7	1111	IRQ15
Bits	Mapping	Bits	Mapping																																				
0000	Reserved	1000	Reserved																																				
0001	Reserved	1001	IRQ9																																				
0010	Reserved	1010	IRQ10																																				
0011	IRQ3	1011	IRQ11																																				
0100	IRQ4	1100	IRQ12																																				
0101	IRQ5	1101	Reserved																																				
0110	IRQ6	1110	IRQ14																																				
0111	IRQ7	1111	IRQ15																																				

ITSS has registers for each IRQ to change the polarity (CR offsets 3200h – 320Fh), default value as below:

- IRQ [16-23] Polarity defaults to Active Low
- IRQ [0-15, 24-119] Polarity defaults to Active High

Bit	Type	Reset	Description
31:0	RW	00FF0000	IRQ 31-0 Active High Polarity Disable (IPCO.IRQxAHPOLDIS): When set to 1, the interrupt polarity associated with IRQ31 down to IRQ0 is inverted to appear as Active Low to IOAPIC. When set to 0, the interrupt appears as Active High to IOAPIC.

In APIC mode, PCI interrupts [A:H] are directly mapped to IRQ[16:23].

10.1.4 SoC Devices Interrupt Routing Capabilities

Device	SB EP Port ID (Hex)	Interrupt Supported?	Dev.	Fun.	Interrupts	Comments
DPTF (Camarillo, Punit)	46	Yes	0	1	Assert/Deassert_IRQn	No MSI support
NPK	88	Yes	0	2	Assert/Deassert_INT[A-D]	supports MSI
Gen	30	Yes	2	0	Assert/Deassert_INTA	supports MSI (aka display)
Iunit	32	Yes	3	0	Assert/Deassert_INTA	supports MSI
PMC	82	Yes	13	1	Assert/Deassert_IRQn	*IRQn - PMC SCI routed as interrupt as well as on behalf of other IPs for S0ix wakes
SPI	93	no	13	2	N/A	
shared SRAM		no	13	3	N/A	
Audio	92	Yes	14	0	Assert/Deassert_IRQn Assert/Deassert_INT[A-D]	supports MSI
TXE-HECI1		Yes	15	0	Assert/Deassert_INT[A-D]	supports MSI
TXE-HECI2	97	Yes	15	1	Assert/Deassert_INT[A-D]	supports MSI
TXE-HECI3	97	Yes	15	2	Assert/Deassert_INT[A-D]	supports MSI
ISH	94	Yes	17	0	Assert/Deassert_IRQn	
USB-Host (xHCI)	A2	Yes	21	0	Assert/Deassert_INTA	supports MSI
USB-Device (xDCI)	A4	Yes	21	1	Assert/Deassert_IRQn	No MSI support
I2C 0	90	Yes	22	0	Assert/Deassert_IRQn	
I2C 1	90	Yes	22	1	Assert/Deassert_IRQn	



Device	SB EP Port ID (Hex)	Interrupt Supported?	De v.	Fun.	Interrupts	Comments
I2C 2	90	Yes	22	2	Assert/Deassert_IRQn	
I2C 3	90	Yes	22	3	Assert/Deassert_IRQn	
I2C 4	90	Yes	23	0	Assert/Deassert_IRQn	
I2C 5	90	Yes	23	1	Assert/Deassert_IRQn	
I2C 6	90	Yes	23	2	Assert/Deassert_IRQn	
I2C 7	90	Yes	23	3	Assert/Deassert_IRQn	
UART 0	90	Yes	24	0	Assert/Deassert_IRQn	
UART 1	90	Yes	24	1	Assert/Deassert_IRQn	
UART 2	90	Yes	24	2	Assert/Deassert_IRQn	
UART 3	90	Yes	24	3	Assert/Deassert_IRQn	
SPI 0	90	Yes	25	0	Assert/Deassert_IRQn	
SPI 1	90	Yes	25	1	Assert/Deassert_IRQn	
SPI 2	90	Yes	25	2	Assert/Deassert_IRQn	
PWM	82	no	26	0	N/A	
SD Card	D6	Yes	27	0	Assert/Deassert_IRQn	
eMMC	D6	Yes	28	0	Assert/Deassert_IRQn	
UFS	D6	Yes	29	0	Assert/Deassert_IRQn	
SDIO	D6	Yes	30	0	Assert/Deassert_IRQn	

10.1.5 Additional Requirement for IA FW

1. IA FW to configure which INTx pin is used by an internal PCI function to drive its interrupt request, and which input pin (PIROQy) on the interrupt router this INTx is mapped to. In other words, a PCH internal PCI function capable of generating an interrupt can be configured to drive any one of the INTA#, INTB#, INTC# or



INTD# pins, which in turn can be mapped to any one of the 8 input pins of the interrupt router (i.e., PIRQA, PIRQB, PIRQC, PIRQD, PIRQE, PIRQF, PIRQG, and PIRQH).

- For a single-function PCI device capable of generating an interrupt, use INTA.
 - For a multi-function device, at least one of the interrupt-capable functions must use INTA.
 - Always include an entry for INTA when reporting device interrupt routing information of a multi-function device to OS, even though the function using INTA pin may actually be “hidden” by the Function Disable register.
2. In S0, UART and SD card would normally send assert_IRQn message to ITSS. In S0ix, UART and SD Card interrupts are not detected by their respective controllers. PMC will send the interrupt to A/PIC on behalf of the device. IA FW must program the IRQ value for UART and SD card in device and in PMC.

10.1.6 Reporting Interrupt Routing to OS

The PCI interrupt routing scheme for all motherboard and chipset internal devices and PCI/PCIe slots behind the root ports is platform specific and needs to be reported to the OS by IA FW via the following interfaces:

- PCI BIOS INT1Ah Service, Sub-function B1xxh (PCI BIOS Specification, non-ACPI environment)
- “\$PIR” PCI IRQ Routing Table (Microsoft* specification, non-ACPI environment).
- Multi-Processor Table (MPS specification, non-ACPI environment)
- ACPI _PRT packages (ACPI Specification, ACPI environment)
- ACPI Multiple APIC Description Table (ACPI Specification, ACPI environment)
- Refer the above specifications for detailed formats of interrupt routing tables and data structures.

10.1.6.1 Example PRT Packages for SoC Interrupt Routing

The following is an ASL code example to illustrate interrupt routing on an Apollo Lake platform, with _PRT packages for PIC mode only.

```

//-----
-----
// Apollo Lake SoC INTERRUPT ROUTING _PRT EXAMPLE (PIC MODE)
//-----
-----

Scope(\_SB) {

    Device(PCIO) {
        bridge
        Name(_HID, EISAID("PNP0A03")) // PnP ID for PCI Bus
    }
}

```



```
Name(_ADR, 0x00000000)
Name(_BBN, 0x0000) // Bus number, optional for the Root PCI Bus

Name(_PRT, Package()){

// SD Host #0 - eMMC
    Package(){0x0010FFFF, 0, LNKA, 0 },

// SD Host #1 - SDIO
    Package(){0x0011FFFF, 0, LNKB, 0 },

// SD Host #2 - SD Card
    Package(){0x0012FFFF, 0, LNKC, 0 },

// SATA Controller
    Package(){0x0013FFFF, 0, LNKD, 0 },
// xHCI Host
    Package(){0x0014FFFF, 0, LNKE, 0 },

// Low Power Audio Engine
    Package(){0x0015FFFF, 0, LNKF, 0 },


// USB OTG
    Package(){0x0016FFFF, 0, LNKG, 0 },


// SIO2 DMA
// SIO2 I2C #4
    Package(){0x0018FFFF, 0, LNKB, 0 },
// SIO2 I2C #1
// SIO2 I2C #5
    Package(){0x0018FFFF, 2, LNKD, 0 },
// SIO2 I2C #2
// SIO2 I2C #6
    Package(){0x0018FFFF, 3, LNKC, 0 },
// SIO2 I2C #3
// SIO2 I2C #7
    Package(){0x0018FFFF, 1, LNKA, 0 },
// GbE Controller
    Package(){0x0019FFFF, 0, LNKE, 0 },
// TXE
    Package(){0x001AFFFF, 0, LNKF, 0 },
// Intel® High Definition Audio (Intel® HD Audio) Controller
    Package(){0x001BFFFF, 0, LNKG, 0 },


// Host Bridge
// Mobile IGFX
```



```

        Package(){0x0002FFFF, 0, LNKA, 0 },
    }) // end _PRT

Device(PC11) {    // DMI to PCI Bridge
    Name(_ADR, 0x001E0000) // Device 30
    Name(_PRT, Package()){
        // PCI Slot 1
        Package(){0x0000FFFF, 0, LNKF, 0 },
        Package(){0x0000FFFF, 1, LNKG, 0 },
        Package(){0x0000FFFF, 2, LNKH, 0 },
        Package(){0x0000FFFF, 3, LNKE, 0 },
        // PCI Slot 2
        Package(){0x0001FFFF, 0, LNKG, 0 },
        Package(){0x0001FFFF, 1, LNKF, 0 },
        Package(){0x0001FFFF, 2, LNKE, 0 },
        Package(){0x0001FFFF, 3, LNKH, 0 },
        // PCI Slot 3
        Package(){0x0002FFFF, 0, LNKC, 0 },
        Package(){0x0002FFFF, 1, LNKD, 0 },
        Package(){0x0002FFFF, 2, LNKB, 0 },
        Package(){0x0002FFFF, 3, LNKA, 0 },
        // PCI Slot 4
        Package(){0x0003FFFF, 0, LNKD, 0 },
        Package(){0x0003FFFF, 1, LNKC, 0 },
        Package(){0x0003FFFF, 2, LNKF, 0 },
        Package(){0x0003FFFF, 3, LNKG, 0 },
    }) // end _PRT
} // end PC11

Device(PEX1) {
    Name(_ADR, 0x001C0000) // PCI Express Root Port1
    Name(_PRT, Package()){
        // PCI Express Slot1
        Package(){0x0000FFFF, 0, LNKA, 0 },
        Package(){0x0000FFFF, 1, LNKB, 0 },
        Package(){0x0000FFFF, 2, LNKC, 0 },
        Package(){0x0000FFFF, 3, LNKD, 0 },
    })
} // end PEX1

Device(PEX2) {
    Name(_ADR, 0x001C0001) // PCI Express Root Port2
    Name(_PRT, Package()){
        // PCI Express Slot1
        Package(){0x0000FFFF, 0, LNKB, 0 },
        Package(){0x0000FFFF, 1, LNKC, 0 },
    }
}

```



```
        Package(){0x0000FFFF, 2, LNKD, 0 },
        Package(){0x0000FFFF, 3, LNKA, 0 },
    })
} // end PEX2

Device(PEX3) {
Name(_ADR, 0x001C0002) // PCI Express Root Port3
Name(_PRT, Package()){
// PCI Express Slot3
    Package(){0x0000FFFF, 0, LNKC, 0 },
    Package(){0x0000FFFF, 1, LNKD, 0 },
    Package(){0x0000FFFF, 2, LNKA, 0 },
    Package(){0x0000FFFF, 3, LNKB, 0 },
}
} // end PEX3

Device(PEX4) {
Name(_ADR, 0x001C0003) // PCI Express Root Port4
Name(_PRT, Package()){
// PCI Express Slot4
    Package(){0x0000FFFF, 0, LNKD, 0 },
    Package(){0x0000FFFF, 1, LNKA, 0 },
    Package(){0x0000FFFF, 2, LNKB, 0 },
    Package(){0x0000FFFF, 3, LNKC, 0 },
}
} // end PEX4

} // end PCI0} // end _SB scope
```

10.2 Processor Interrupt Support

The IA FW boot flow is described follows:

The IA FW will configure the IOxAPIC for Processor Interrupt delivery by setting the registers as described in the previous section.

- The IA FW will build the MP and/or ACPI APIC tables for the OS.
- Control is passed to the boot strap loader to handle control to the OS.
- OS will switch the interrupt operation from PIC or virtual wire mode to symmetric I/O Mode (IOAPIC usage).
- When the OS switches the mode of interrupt delivery to the IOxAPIC the interrupts are delivered using the PSB Interrupt mechanism.



10.2.1 Processor Interrupt Delivery Mechanism

APL SoC supports the delivery of interrupts to the processor by using one of the two mechanisms:

- Interrupt delivery by using the INTR VLW to the processor, when IOxAPIC is disabled.
- Interrupt delivery from the IOxAPIC by sending interrupt messages (MSI) when IOxAPIC is enabled.

10.2.1.1 Configuration of the IOxAPIC

The IOxAPIC is accessed via an indirect addressing scheme. These registers are mapped into memory space. The registers are shown below.

Table 9-1. IOxAPIC Registers

Address	Symbol	Register
FEC00000h	IDX	Index Register
FEC00010h	WDW	Window Register
FEC00040h	EOI	EOI Register

- IDX – Index Register: selects which indirect register appears in the window register to be manipulated by software. Software will program this register to select the desired APIC internal register.
- WDW – Window Register specifies the data to be read or written to the register pointed to by the IDX register. This register can be accessed only in DW quantities.

The requirements for IA FW to enable IOxAPIC are:

1. Set the AE bit, in P2SB MMIO offset 0x64[8], to '1b'. Read back the value written. Where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10. The rest of detailed procedures are in ScIoApicInit in SIC reference package.

Build the MP table and/or ACPI APIC table for the OS.

Maximum Redirection Entries (MRE) in APIC Version Register (VER), offset 0x01, [23:16] can be read to indicate number of interrupts. Current value is 86 (56h), enabling 87 interrupt vectors.

The interrupt routing table is located at address 0x10 and on, 64-bit entries. These entries have the following format:

Table 9-2. IOxAPIC Interrupt Routing Table

Bits	Type	Description
63:56	RW	Destination ID (DID): Destination ID of the local APIC.
55:48	RW	Extended Destination ID (EDID): Extended destination ID of the local APIC.
47:17	RO	Reserved



Bits	Type	Description
16	RW	Mask (MSK): When set, interrupts are not delivered nor held pending. When cleared, and edge or level on this interrupt results in the delivery of the interrupt.
15	RW	Trigger Mode (TM): When cleared, the interrupt is edge sensitive. When set, the interrupt is level sensitive.
14	RW	Remote IRR (RIRR): This is used for level triggered interrupts its meaning is undefined for edge triggered interrupts. This bit is set when IOxAPIC sends the level interrupt message to the CPU. This bit is cleared when an EOI message is received that matches the VCT field. This bit is never set for SMI, NMI, INIT, or ExtINT delivery modes.
13	RW	Polarity (POL): This specifies the polarity of each interrupt input. When cleared, the signal is active high. When set, the signal is active low.
12	RO	Delivery Status (DS): This field contains the current status of the delivery of this interrupt. When set, an interrupt is pending and not yet delivered. When cleared, there is no activity for this entry
11	RW	Destination Mode (DSM): This field is used by the local APIC to determine whether it is the destination of the message.
10:08	RW	Delivery Mode (DLM): This field specifies how the APICs listed in the destination field should act upon reception of this signal. Certain Delivery Modes will only operate as intended when used in conjunction with a specific trigger mode. These encodings are: 000 Fixed 001 Lowest Priority 010 SMI – Not supported. <i>011 Reserved</i> 100 NMI – Not supported. 101 INIT – Not supported. <i>110 Reserved</i> 111 ExtINT
07:00	RW	Vector (VCT): This field contains the interrupt vector for this interrupt. Values range between 10h and FEh.

10.2.1.2 Steps Involved in Delivering the Interrupt

These are the steps involved in the delivering of a processor interrupt.

1. IOAPIC detects that an interrupt event has happened and sets the IRR bit associated with that interrupt.

IOAPIC checks whether the IOAPIC interrupt is to be delivered by the Serial Bus mechanism or the Processor Interrupt mechanism. If the Interrupt is to be delivered by the interrupt messages through address remapping mechanism SoC follows Step 3 and 4.

SoC automatically flushes all the upstream buffers.

IOAPIC delivers the interrupt in the form of a 32-bit memory write cycle with a known format for address and data as defined in "Interrupt Message Address" and "Interrupt Message Data".

**Table 9-3. Interrupt Message Address Format**

Bits	Description
Bits 31:20	FEEh
Bits 19:12 (Destination ID)	Same as bits 63:56 of I/O Redirection Table Entry
Bits 11:4 (Extended Destination ID)	Same as bits 55:48 of I/O Redirection Table Entry
Bit 3 (Redirect Hint)	0/1
Bit2 (Destination Mode)	0/1 (based on bit 3)
Bits 1:0	00b

Table 9-4. Interrupt Message Data Format

Bits	Description
Bits 31:16	0000h
Bit 15 (Trigger Mode)	0/1 (Edge/Level)
Bit 14 (Delivery Status)	0/1 (Deassert / Assert)
Bits 13:12	00b
Bit 11 (Destination Mode)	0/1 (Physical /Logical)
Bits 10:8 (Delivery Mode)	Same as 10:8 of I/O Redirection Table entry
Bits 7:0 (Vector)	

10.2.1.3 IOxAPIC Input Mapping Table

The following table describes the APIC interrupt mapping. In Apollo Lake, there are 120 IRQs in total.

Table 9-5. IOxAPIC Input Mapping

IO APIC Pin	Routing
0	8259
1	KBD emulation
2	8254 Counter 0, HPET #0 (legacy mode)
3	SDCard
4	uart0
5	uart1
6	uart2
7	uart3
8	RTC, HPET #1 (legacy mode)



IO APIC Pin	Routing
9	Option for configurable sources including internal ACPI/PCI devices, SCI and TCO
10	Option for configurable sources including internal ACPI/PCI devices, SCI and TCO
11	Option for configurable sources including internal ACPI/PCI devices, SCI, TCO, HPET #2
12	Option for configurable sources including internal ACPI/PCI devices, HPET#3
13	xDCI
14	Option for configurable sources including GPIO, internal ACPI/PCI devices
15	Option for configurable sources including internal ACPI/PCI devices
16	Option for configurable sources including internal PIRQA, internal ACPI/PCI devices
17	PIROB (xHCI)
18	Option for configurable sources including internal PIRQC, internal ACPI/PCI devices
19	Option for configurable sources including internal PIRQD, internal ACPI/PCI devices
20	Option for configurable sources including internal PIRQE, SCI, TCO, internal ACPI/PCI devices and HPET
21	Option for configurable sources including internal PIRQF, SCI, TCO, internal ACPI/PCI devices and HPET
22	Option for configurable sources including internal PIRQG, SCI, TCO, internal ACPI/PCI devices and HPET
23	Option for configurable sources including internal PIRQH, SCI, TCO, internal ACPI/PCI devices and HPET
24	Punit (DPTF)
25	Audio
26	ISH
27	I2C 0
28	I2C 1
29	I2C 2
30	I2C 3

IO APIC Pin	Routing
31	I2C 4
32	I2C 5
33	I2C 6
34	I2C 7
35	SPI 0
36	SPI 1
37	SPI 2
38	UFS
39	EMMC
40	PMC IPC
41	PMC PMIC
42	SDIO
43	reserved
44	reserved
45	reserved
46	reserved
47	reserved
48	reserved
49	reserved
50 - 119	GPIO

10.2.2 SoC Settings to Generate NMI

Since APL SoC PCU is the only agent in the system that delivers the NMI to the CPU, all the sources and requests for generation of NMI have to come into PCU. The sources for NMI are:

1. SERR# assertion
2. IOCHK signal (Serial IRQ bus only)
3. NMI pin
4. NMI_NOW bit in P2SB MMIO + 0xD03330 [0], Where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10

Software reads the NMI Status (P2SB MMIO + 0xD03330) determine the NMI source. After the NMI interrupt routine processes the interrupt, software clears the NMI status bits, refer Apollo Lake SoC EDS for the complete definition of the NMI Status and Control Register and the GPIO NMI Status and Control register.



The NMI Enable bit in the NMI Enable / RTC index register (I/O port 0x70, bit [7]) can be used to mask the NMI VLW and disable/enable all NMI sources.

Refer Apollo Lake SoC EDS for the complete definition of the NMI Status and Control Register as well as the NMI Enable Register.

10.2.3 NMI Handling in OS Environment

APL SoC is capable of generating NMI's from various sources. Many of these NMI sources are currently not handled by the OS environment. In case the IA FW enables NMI sources that should be handled by the OS, it should be handled in the SMI handler of the IA FW by routing the NMI to generate an SMI. The routing of NMI to SMI is possible by setting the NMI2SMI_EN bit in the NMI Status and Control register (P2SB MMIO + 0xD03330[2], where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10).

If the system designer implements the watchdog timer function, the IA FW should generate an NMI after the first timeout to enable an OS crash dump function to occur on a system hang.

10.2.4 Routing NMI to SMI

When NMI is routed to an SMI, i.e. the NMI2SMI_EN bit (P2SB MMIO + 0xD03330[2], where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10) is set, the IA FW should add the following functionality to the SMI# handler:

1. Check if the source of the SMI# was iLB by checking if the ilb_smi_sts bit in the SMI_STS Register, ACPI Base address(traditionally it's io port 0x400) +0x34[15] = 1'b1. If this bit is not set, then continue with normal SMI# processing and do not execute any of the steps below.
2. Check if the SMI source was routed from NMI by checking NMI2SMI_STS bit (P2SB MMIO + 0xD03330[3], where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10). If this bit is not set, then continue with normal TCO SMI# processing.
3. Search for the NMI source by reading I/O port 0x61 and the NMI Status and Control register (P2SB MMIO + 0xD03330, where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10).
4. Perform actions that are required by NMI cause.
5. If real NMI is needed, generate an NMI by doing the following steps:
 - a) Read the NMI2SMI_EN bit (P2SB MMIO + 0xD03330[2], where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10), save it for future restore.
 - b) Clear the NMI2SMI_EN bit (P2SB MMIO + 0xD03330[2], where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10).
 - c) Enable NMI by setting the NMI_EN bit (IO port 0x70[7]) = 1'b0.
6. IO port 0x70 status must be restored prior to exit from SMI.
 - a) Set the NMI_NOW bit (P2SB MMIO + 0xD03330[0], where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10) = 1'b1.



- b) Clear the NMI_NOW bit. To do this, write a 1'b1 to NMI_NOW (P2SB MMIO + 0xD03330[0], where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10).
 - c) Restore the NMI2SMI_EN bit (P2SB MMIO + 0xD03330[2], where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10) saved in earlier step.
7. Clear the NMI cause status bit.
- a) If SERR#_NMI_STS (I/O port 0x61[7]) is set: disable PCI_SERR_EN (I/O port 0x61[2] = 1'b0) and then restore its previous value
 - b) If IOCHK_NMI_STS (I/O port 0x61[6]) is set: disable IOCHK_NMI_EN (I/O port 0x61[1] = 1'b0) and then restore its previous value
 - c) If NMI_NOW_STS (P2SB MMIO + 0xD03330[1], where P2SB MMIO is at Dev:0x0d, Fun:0, Reg:0x10) is set: write 1'b1 to the status bit

Continue with TCO SMI# processing.

10.2.5 SoC Settings to Generate SMI

In order to enable SMI generation the GLB_SMI_EN bit (ACPIBASE+0x40[0]) should be set. In addition software should enable the required SMI sources by setting the right bit in the SMI_EN register (ACPIBASE+0x40):

Table 9-6. SMI_EN Register - bit Definition

Bit #	Name	Short Description
0	GBL_SMI_EN	Enables the generation of SMIs in the system upon any enabled SMI event
1	EOS	SMI handler sets this bit when it finishes handling the SMI
2	BIOS_EN	Enables the generation of SMI when ACPI software writes a '1b' to the GBL_RLS bit
3	Reserved	Reserved
4	SMI_ON_SLP_EN	If this bit is set, the Intel Apollo Lake SoC will generate an SMI when a write access attempts to set the SLP_EN bit (in the PM1_CNT register). Furthermore, the Intel Apollo Lake SoC will not put the system to a sleep state. This allows the SMI# handler work around chip-level bugs. It is expected that the SMI handler will turn off the SMI_ON_SLP_EN bit before actually setting the SLP_EN bit.
5	APMC_EN	If set, this enables writes to the APM_CNT register to cause an SMI#
6	SWSMI_TMR_EN	Software sets this bit to a 1 to start the Software SMI# Timer
7	BIOS_RLS	Enables the generation of an SCI interrupt for ACPI software when a '1b' is written to this bit position by IA FW software
12:8	Reserved	Reserved
13	TCO_EN	Enables the TCO logic to generate SMI
14	PERIODIC_EN	Enables the generation of SMI according to periodic SMI timer



Bit #	Name	Short Description
16:15	Reserved	Reserved
17	USB_SMI_EN	Enables SMI from USB
18	USB_IS_SMI_EN	Enables Intel-Specific USB2 SMI logic to cause SMI
26:19	Reserved	Reserved
27	GPIO_SMI_EN	Enables SMI from GPIO
31:28	Reserved	Reserved

It is recommended to use SMI_LOCK bit to prevent SMI sensitive configuration bits from being changed by malicious code.

10.2.6 SMI LOCK Bit

When the SMI_LOCK bit, PBASE+0x24[4], is set, writes to the GBL_SMI_EN bit (ABASE+0x40[0]) for enabling and disabling SMI generation have no effect. The SMI_LOCK bit can only be cleared by PLTRST#.

10.2.7 Setting EOS Bit

Setting the EOS bit, ACPIBASE+0x40[1], will de-assert the SMI virtual signal and will re-arm SMIs. If there is a pending and enabled SMI status bit, the EOS bit will read back as cleared. In this case, an SMI will occur as soon as the resume instruction is executed.

10.2.8 SMI Status Bits

The SMI status bits must be cleared by software after the SMI source has been de-asserted. It is up to the SMI handler to de-assert the SMI source and clear the status bits. Status bits that are set for SMI sources can be set even if they are not enabled and do not need to be serviced or cleared. SMI Status bits should be qualified with their respective enables before being serviced. If enabled status bits are not cleared, the EOS bit will remain set.

10.2.9 IO Ports 0xB2 / 0xB3

OS and IA FW are using IO ports 0xB2 and 0xB3 registers to initiate SyncSMI and pass data to SMI handler. These IO ports include their enable (APMC_EN) and status (APM_STS) bits.

10.2.10 Advanced Power Management Control Port (APM_CNT)

IO fix address	0xB2
Size	8 bits
Default	0x00



Bits	Access Type	Default	Description
7:0	RW	0x00	Advanced Power Management Control Port (APM_CNT) Used to pass an APM command between the OS and the SMI handler. Writes to this port not only store data in the APMC register, but also generates an SMI# when the APMC_EN bit is set.

10.2.11 Advanced Power Management Status Port (APM_STS)

IO fix address 0xB3
Size 8 bits
Default 0x00

Bits	Access Type	Default	Description
7:0	RW	0x00	Advanced Power Management Status Port (APM_STS) Used to pass data between the OS and the SMI handler. Basically, this is a scratchpad register and is not affected by any other register or function (other than a PCI reset).

10.3 SCI Routing

Interrupt routing for System Control Interrupt (SCI) is essential for operating system to trigger hardware power management events successfully.

Interrupt routing for SCI is disabled by default. IA FW is required to route SCI to IRQ9 by program "SCI IRQ Select" bits in "ACPI Control" register, IBASE + 0x00 [2:0] to 000b.

§



11 **UART**

APL SoC Implements four independent UART interface: UART0, UART1, UART2 and UART3 respectively.

11.1 **UART PCI Addresses**

IA FW should configuration UART as PCI mode.

Table 10-1. UART Devices PCI BAR Register

Device / Function	Device Name	Register
Device 24 / Function 0	UART0	0x10
Device 24 / Function 1	UART1	0x10
Device 24 / Function 2	UART2	0x10
Device 24 / Function 3	UART3	0x10

11.2 **UART ACPI Requirement**

The IA FW must expose UART device as PCI device in ACPI.

Table 10-2. Describes ACPI ASL Code for UART: APL UART ACPI ASL Sample Code

```
//  
// LPIO1 HS-UART #1  
//  
Device(URT1) {  
    Name (_ADR, 0x00180000)  
    Name (_DDN, "Intel(R) HS-UART Controller #1 - 80860ABC")  
  
    Name (RBUF, ResourceTemplate ()  
    {  
        //FixedDMA(0x2, 0x2, Width32Bit, )  
        //FixedDMA(0x3, 0x3, Width32Bit, )  
    })  
    Method (_CRS, 0x0, NotSerialized)  
    {  
        Return (RBUF)  
    }  
} // Device (URT1)
```

§



12 xHCI Support

APL SoC implements an xHCI host controller (B0:D21:F0) which supports up to 6 (1x USB Dual Role, 1 dedicated port, 3x multiplexed with PCIe* 2.0, 1x multiplexed with SATA 3.0), 1 SSIC port.

12.1 xHCI Controller Options in Reference Code

APL SoC integrates xHCI controller that enumerated on PCI. The xHCI controller is responsible for enumerating USB 2.0, USB 3.0 and SSIC buses (under OS control).

12.2 xHCI Controller Initialization

The IA FW performs low level chipset initialization of the xHCI controller. The following steps related to xHCI are performed:

- Enabling or disabling xHCI controller on the PCI bus.
- IA FW initializes xHCI registers settings.
- PCI enumeration is performed on the xHCI controller.
- IA FW to describe USB ports by using _UPC and _PLD method call (ACPI).

12.2.1 Function Disabling xHCI Controller

In some instances, the IA FW may choose to disable the xHCI controller even though the xHCI controller is enabled in fuse and soft strap. To disable a host controller, the IA FW must first place the xHCI controller in RTD3Hot state by program PM_CS_PS, D21:F0:0x74 [1:0] = 11b and then set the Function Disable register for xHCI at PBASE + 0x34[1] to disable the xHCI controller.

12.2.2 Port Disable Override

In Apollo Lake, Port disable override on xHCI ports is implemented and BIOS could set via USB MMIO 0x84F8 and 0x84FC

12.2.3 MPHY Initialization

IA FW should perform the following private configuration space programming sequence for USB High Speed IO through MsgBus. 1st element is MsgBus Port ID, 2nd element is for masking the read register value and the 3rd element is the value attempt to be written into the register. The 4th element is register offset address. The 5th element is for Lane Phy mode. For the MsgBus programming approach, refer Section 1.8 of Volume 1 for the details. For detail setting, refer SIC Release Package for detail.



Table 11-1. MPHY Init Sample Code

```
{0xA5, 0xFFFFFFFF, 0x20000000, 0x8020, SC_LANE_OWN_COMMON},  
//LANE  
{0xB1, 0xFFFFFFFF, 0x20000000, 0x8020, SC_LANE_OWN_COMMON},  
//LANE  
{0xB1, 0xFFFFFFF0F, 0xC0, 0x110, V_SC_PCR_FIA_LANE_OWN_SATA},  
//UNICAST  
{0xA5, 0xFFFFFFF00, 0x40, 0x8034, SC_LANE_OWN_COMMON},  
//LANE  
{0xA5, 0xFFE0E0E0, 0xC0C0F, 0x24, V_SC_PCR_FIA_LANE_OWN_USB3},  
//UNICAST  
{0xA5, 0xFFFFFFF9F, 0x60, 0x28, V_SC_PCR_FIA_LANE_OWN_PCIE},  
//UNICAST  
{0xA5, 0xFFFFFFF9F, 0x60, 0x28, V_SC_PCR_FIA_LANE_OWN_USB3},  
//UNICAST  
{0xB1, 0xFFFFFFF9F, 0x60, 0x28, V_SC_PCR_FIA_LANE_OWN_SATA},  
//UNICAST  
{0xA5, 0xFDFF00FF, 0x2001C00, 0x04, V_SC_PCR_FIA_LANE_OWN_PCIE},  
//UNICAST
```

12.2.4 Locking xHCI Register Settings

APL includes xHCI registers lock-down feature, which allows some xHCI registers to be locked down and Read-Only (RO) when the xHCI lockdown bits are enabled.

After xHCI is initialized, IA FW should lock the xHCI configuration registers to read only. This prevents any unintended changes. There is also a lockdown feature for over-current registers. IA FW should set these bits to lock down the settings prior to end of POST.

1. Set Access Control bit at D21:F0:0x40 [31] (XHCC1.ACCTRL) to '1b' to lock xHCI register settings.
2. Set OC Configuration Done bit at D20:F0:0x44 [31] (XHCC2.OCCFDONE) to '1b' to lock overcurrent mappings from further changes.

12.2.5 xHCI Legacy SMI Enabling

In order to enable xHCI Legacy SMI, take the following steps:

1. Set ACPIBASE + 0x30 [31] = 1b

Set USB Legacy Support Capability + 0x04 [0] (USBLEGCTLSTS.USBSMIE) = 1b. In APL, the USB Legacy Support Capability is located at xHCIBAR + 0x846C.

12.2.6 Additional Settings for xHCI Controller

1. Program HCSPARAMS3.U2DEL, xHCIBAR + 0x0C [31:16] = 0x200.
2. Program HCSPARAMS3.U1DEL, xHCIBAR + 0x0C [7:0] = 0x0A.
3. Set XHCC2.DREQBCC, D21:F0:0x44 [25].
4. Program XHCC2.WRREQSZCTRL, D21:F0:0x44 [24:22] = 111b.
5. Program XHCC2.UNPPA, D21:F0:0x44 [19:14] = 0x3F.
6. Set XHCC2.RAWDD, D21:F0:0x44 [11].



7. Set XHCC2.WAWDE, D21:F0:0x44 [10].
8. Program XHCC2.SWACXIHB, D21:F0:0x44 [9:8] = 10b.
9. Program XHCC2.SWADMIL1IHB, D21:F0:0x44 [7:6] = 10b.
10. Program XHCC2.RDREQSZCTRL, D21:F0:0x44 [2:0] = 111b.
11. Set HCCPARAMS.SEC, xHCIBAR + 0x10 [10].
12. Set HCCPARAMS.SPC, xHCIBAR + 0x10 [9].
13. Clear HCCPARAMS.LHRC, xHCIBAR + 0x10 [5].
14. Set xHCIBAR + 0x8094 [23].
15. Set xHCIBAR + 0x8094 [21].
16. Set xHCIBAR + 0x8094 [14].
17. Set xHCIBAR + 0x80B0 [24].
18. Clear USB2_LINK_MGR_CTRL_REG1.L1_EXIT_RECOVERY_MODE, xHCIBAR + 0x80F0 [20].
19. Set xHCIBAR + 0x80FC [25].
20. Set HOST_CTRL_TRM_REG2.DIS_CPL_CHK_DEV_CRDT, xHCIBAR + 0x8110 [20].
21. Set HOST_CTRL_TRM_REG2.EN_TDCDS, xHCIBAR + 0x8110 [11].
22. Clear HOST_CTRL_TRM_REG2.CLR_CPCR, xHCIBAR + 0x8110 [2].
23. Set xHCIBAR + 0x8154 [31].
24. Set XLTP_LTV1.XLTRE, xHCIBAR + 0x8174 [24].
25. Program XLTP_LTV1.PA_LTV, xHCIBAR + 0x8174 [11:0] = 0xC0A.
26. Program XLTP_HITC.MHIT, xHCIBAR + 0x817C [28:16] = 0x0332.
27. Program XLTP_HITC.HIWL, xHCIBAR + 0x817C [12:0] = 0x00A3.
28. Program XLTP_MITC.MMIT, xHCIBAR + 0x8180 [28:16] = 0x00CB.
29. Program XLTP_MITC.MIWL, xHCIBAR + 0x8180 [12:0] = 0x0028.
30. Program XLTP_LITC.MLIT, xHCIBAR + 0x8184 [28:16] = 0x0064.
31. Program XLTP_LITC.LIWL, xHCIBAR + 0x8184 [12:0] = 0x001E.
32. Set XECP_CMDM_CTRL_REG1.FCE, xHCIBAR + 0x818C [20].
33. Set XECP_CMDM_CTRL_REG1.CSS_EN, xHCIBAR + 0x818C [16].
34. Clear XECP_CMDM_CTRL_REG1.CC_ES_EN, xHCIBAR + 0x818C [8].
35. Set xHCIBAR + 0x8194 [25].
36. Perform the Low Power Feature programming for xHCI
37. Program xHCIBAR + 0x890C [27:0] = 0x033C000.
38. Program xHCIBAR + 0x8A1C [27:0] = 0x033C000.
39. Program xHCIBAR + 0x8910 [27:0] = 0x00FC000.
40. Program xHCIBAR + 0x8A20 [27:0] = 0x0A3C000. Optional for G1
41. Program xHCIBAR + 0x8914 [27:0] = 0x092C000.
42. Program xHCIBAR + 0x8A24 [27:0] = 0x092C000.
43. Set xHCIBAR + 0x8904 [25] = 1b.
44. Set xHCIBAR + 0x8A04 [25] = 1b.
45. Set xHCIBAR + 0x8904 [23] = 1b.
46. Set xHCIBAR + 0x8A04 [23] = 1b.
47. Set xHCIBAR + 0x880C [30] = 1b and xHCIBAR + 0x883C [30] = 1b once step 37 to step 46 are done.



48. Set xHCIBAR + 0x880C [31] = 1b if SSIC port 1 is unused.
49. Set xHCIBAR + 0x883C [31] = 1b if SSIC port 2 is unused.
50. Set XHCI_MEM_DUAL_ROLE_CFG0 = 0x1310800
51. Set xHCI MSI_NEXT to 0x90 in XHCI_MSI_CAPID

12.2.7 Additional Programming Requirements during USB Initialization

IA FW must execute the following steps as part of IA FW initialization of the USB controller on both cold boot (G3/S5) and S3/S4 resume path.

- Set "PME_En" bit of PM_CS register, D21:F0:0x74 [8] = 1b.
- USB2 L1 is enabled and exposed risk as the required workaround in APL do not honor the negotiated exit time for L1 state, hence causing the risk of data loss in the USB link and therefore HLC bit must be cleared to disable USB2 L1 for APL SoC, 0x8008[19]=0.

12.3 Debug Port Support

In order to meet WHQL requirements at least one exposed port needs to support Debug Capability. The Debug port needs to be routed to a walk up connector.

The debug capability is supported and visible through the extended capabilities pointer. It is enabled by the debug driver that enables the debug capability through Debug Capability Enable (DCE). The enabling is handled by the debug target's debug capability driver. The debug capability is supported on any port once the debug capability enable bit is set by the debug cap driver. Once the first debug device connection is made no other port will be allowed to be used as a debug device. Refer XHCI specification for additional information.

§



13 USB Device Mode (OTG) Support

In addition to xHCI USB sub-systems shown in previous chapters, APL SoC integrates an USB OTG controller that supports Device-mode operation at Super Speed, High Speed, and Full Speed. The host-mode and OTG discovery/switching functionality of the controller are not used.

The platform may assign a dedicated type B USB socket to this controller, or use ID PIN-based multiplexing to share a type AB socket between this controller and one of the ports of the other USB controllers

13.1 Intel® OTG Initialization Flow

The IA FW initialization flow for the Intel® OTG is as follow:

- IA FW may disable the controller prior to PCI enumeration. If disable then below steps should be skipped.
- IA FW must perform the Snoop programming.
- IA FW must perform the additional programming steps.
- IA FW must enumerate the controller in PCI mode.

13.2 IA FW Disabling Intel® OTG

If Intel® OTG is enabled by fuse but still need to be function disabled, the IA FW can disable the Intel® OTG by following the steps below:

Configure xHCI to static host mode by program xHCI BAR + 0x80D8 [1:0] to 10b.

Poll xHCI BAR + 0x80DC [29] until it reads 1b or else 5ms timeout.

Program xDCI BAR + 0x10F818 [1:0] =2'b11

Poll xDCI BAR + 0x10F810 [11:8] until it reads 1111b or else 5ms timeout

Move Intel® OTG to RTD3Hot by program PMECTRLSTATUS.POWERSTATE, D21:F1:Reg 0x84 [1:0] to 11b.

Set the function disable bit for OTG at PBASE + 0x34 [0].

13.3 Intel® OTG Snoop Programming

System IA FW is required to perform snoop programming on Intel® OTG by setting IOSFCTL.NSNPDIS, offset 0x00 [7] to 1b in OTG private config space. This will prevent non-snooped requests from being issued which would cause memory coherency issues.



13.4 Intel® OTG Additional Programming Steps

Set GUSB2PHYCFG_0.SUSPENDUSB20 bit, OTG BAR0 + 0xC200 [6].

13.5 PCI Mode

Intel® OTG supports PCI mode. In PCI mode the controller is host visible via the PCI configuration space. This is the default mode of operation from S5/S4 exit to S0 entry.

13.5.1 PCI Mode of Operation

Intel® OTG can be a full PCI device at B21:F1, and IA FW can decide to leave Intel® OTG host controller to operate in PCI mode so that operating system can discover the device when doing PCI scan. It is recommended to disable BAR1 by setting PCICFGCTR1.BAR1_DISABLE1, offset 0x500 [7] = 1b in OTG private config space before PCI enumeration.

13.6 Intel® OTG ACPI Requirements

The IA FW must provide ACPI table for each of the Intel® OTG as per ACPI 5.0 specification when OTG is working as an ACPI device. For ACPI ASL definition Refer the SIC reference code. [Table 12-1](#) describes the ACPI ASL code for Intel® OTG.



Table 12-1. Intel® OTG ACPI ASL Sample Code

```

//  

// USB On-The-Go  

//  

Device(OTGA) {  

    Name (_HID, "80865AAA")  

    Name (_CID, "80865AAA")  

    Name (_DDN, "Intel(R) USB OTG Controller - 80865AAA")  

    Name (_UID, 1)  

    Method (_STA, 0x0, NotSerialized)  

    {  

        If (LEqual(OTGD, 0))  

        {  

            Return (0xF) // Enabled  

        }  

        Return (0x0) // Disabled  

    }  

    Method (_CRS, 0x0, NotSerialized)  

    {  

        Name (RBUF, ResourceTemplate ())  

        {  

            Memory32Fixed (ReadWrite, 0x00000000, 0x00001000, BAR0) // MMIO  

1 - OTG MMIO  

            Memory32Fixed (ReadWrite, 0x00000000, 0x00001000, BAR1) // MMIO  

2 - Shadowed PCI Config Space  

            Interrupt (ResourceConsumer, Level, ActiveLow, Exclusive, , , )  

{13} // OTG IRQ  

        })
        Return (RBUF)
    }
} // End "USB On-The-Go"

```

§



14 Integrated SATA Controllers

The APL Serial ATA (SATA) controller supports a 2 SATA3 ports. Refer EDS for details.

SATA controller supports Advanced Host Controller Interface (AHCI). In the following discussion, the term "AHCI Mode" refers to the AHCI programming interface that uses memory-mapped register/buffer space and a command-list-based model.

A separate document, Serial ATA Advanced Host Controller Interface (AHCI) Specification contains details on SATA software configuration and considerations.

14.1 SATA Controller Initialization

The general guidelines for initializing the SATA controller during POST and S3 resume are described in the following sections. Upon resuming from S3, IA FW is responsible for restoring all registers that it initialized during POST.

The high level of SATA Controller initialization flow is as below:

1. Check if SATA Controller is to be enabled
 - a. If SATA Controller is to be disabled, perform disabling steps as indicated in [Section 14.3](#) and exit here.
2. Enable all SATA ports as indicated in [section 14.1.2](#)
3. Program PCS "Port X Enabled" which located at PCI offset 94h[29:24]
4. Perform steps as indicated in [Section 14.1.3](#) to initialize the registers in ABAR.
5. Perform steps as indicated in [Section 14.2](#) for DevSleep enabling.
6. Perform the power optimizing steps as indicated in [Section 14.1.4](#).
7. Check SATA Controller mode, and program the SATA Mode Select as indicated in [Section 14.1.1](#).
8. In later stage (end of DXE), check each SATA port status. Enable a SATA port if any of the condition below is meet:
 - a. A device is attached.
 - b. SATA test mode is enabled.
9. Set MAP "SATA PortX Disable" which located at PCI offset 90h[23:16].
10. Program PCD "Port Clock Disable" at PCI office 90h[7:0].
11. Program SATAGC, SATA PCI offset 9Ch bit[31] to 1'b.

14.1.1 Setting SATA Controller Mode

The IA FW must program the SATA controller mode prior to beginning other initialization steps. The SATA controller mode is set by programming the SATA Mode Select (SMS) field. The IA FW shall never change the SATA controller mode during runtime.



14.1.1.1 AHCI Mode

AHCI mode is selected by programming the SMS field (refer [Section 13.1.1](#) above) to 0b. In this mode, the SATA controller is set up to use the AHCI programming interface. The SATA ports are controlled by a single SATA function, D18:F0. In AHCI mode the Sub Class Code, PCI Offset 0Ah, will be set to 06h. This mode does require specific OS driver support.

14.1.1.2 SATA Mode Setup Option

The IA FW should implement a setup option that provides the user with the ability to select the SATA controller mode. This will ensure that the operating system can be loaded and made operational on the platform if the required device driver support is not available.

14.1.2 Enable Ports

It has been observed that some SATA drives will not start spin-up until the SATA port is enabled by the controller. In order to reduce drive detection time, and hence the total boot time, System BIOS should enable the SATA port early during POST (for example, before memory initialization) by setting the Port x Enable (PxE) bits of the Port Control and Status register, PCI offset 94h. It is important to note that, the "AHCI Enable" is always set. Thus, to initiate spin-up of such drive(s), it is also required to set the PxCmd.SUD of each port, as the CAPS.SSS has a default value of 1. The SCLKCG.PCD bits must be left at 0 before setting the PCS.PxE bits.

Note: If System BIOS intends to clear all PxE bits that are previously '1' to '0' and set back to '1' again, system BIOS should poll (Port x Present) PxP bit being '0' before setting the PxE bit to '1'

In AHCI modes, the IA FW must program SATA PCI offset 94h[n:0] accordingly to ports muxing. In AHCI enabled systems, the PCS register must always be set this way. The status of the port is controlled through AHCI memory space.

Note: If Staggered Spin Up support is desired due to system power load concerns, IA FW should enable one port at a time, poll the Port Present bit of the PCS register and Drive Ready bit of task file status register before enabling the next port. Refer the Serial ATA Advanced Host Controller Interface (AHCI) Specification for details of Staggered Spin Up support. It should be noted that supporting staggered spin up may introduce longer delays.

The ATA/ATAPI-7 specification recommends a 31 second timeout before assuming a device is not functioning properly. Intel recommends implementing a ten second timeout and if the device fails to respond within the first ten seconds, then the IA FW should reinitiate the detection sequence. If the device fails to respond within an additional ten seconds, the IA FW should reinitiate the detection sequence again. If the device fails to respond within ten more seconds, the IA FW can assume the device is not functioning properly.

It is recommended that Software or IA FW clears Serial ATA Error Register (PxSERR) after port reset happens by writing 1 to each implemented bit location.



14.1.3 Initialize Registers in AHCI Memory-Mapped Space

When the SATA controller is configured to operate in AHCI mode, the IA FW must initialize the following memory-mapped AHCI registers specified by ABAR, PCI offset 24h:

- Initialize the registers (CAP, PI, PxCMD) as indicated below:
- CAP register, Host Capabilities register (ABAR + 00h).
 - Set SXS (ABAR + 00h[5]) if External SATA is supported for at least one of the ports
 - Set PSC (ABAR + 00h[13]). This bit informs the Windows* software driver that the AHCI Controller supports the partial low-power state.
 - Set SSC (ABAR + 00h[14]). This bit informs the Windows* software driver that the AHCI Controller supports the slumber low-power state. Set SALP (ABAR + 00h[26]) to enable Link Power Management (LPM) support.
 - Set SCLO (ABAR + 00h[24]) to support Command List Override
 - Set SALP (ABAR + 00h[26]) if SALP is supported to enable Aggressive Link Power Management (LPM) support
 - Set SMPS (ABAR + 00h[28]) if Mechanical Presence Switch is enabled for at least one of the ports
 - Set ISS (ABAR + 00h[20:23]) according to the SATA max speed supported
- Set SSS (ABAR + 00h[27]) to enable SATA controller supports staggered spin-up on its ports, for use in balancing power spikes.

Note: If SATA Test Mode is enabled, then unconditionally clear SSS (ABAR + 00h[27])

Note: PI register, Ports Implemented (ABAR + Ch).

- Set PI (ABAR + Ch) to all 1'b to enable all SATA ports

Note: After IA FW issues the initial write to AHCI Ports Implemented (PI), ABAR + 0Ch, register (after any PLTRST#), BIOS is requested to issue two reads to the AHCI Ports Implemented (PI) register. Furthermore, if BIOS for some reason accesses any of the port specific AHCI address range before setting corresponding PI bit, BIOS is required to read the PI register before the initial write to the PI register.

PxCMD, Port [0-X] Command (ABAR + 118h + (n*80h)) [n = ports number]

- Set SUD, bit[1] to 1'b if the SATA Controller supports staggered spin-up
- Set ALPE, bit[26] to 1'b

Some of the bits in these registers are platform specific and shall be programmed in accordance with the requirements of the platform. The details regarding how these registers shall be programmed can be found in the Serial ATA Advanced Host Controller Interface (AHCI) specification.

Some of the bits in these registers are implemented as read/write-once (R/WO). It is a requirement that the IA FW programs each bit at least once, even if the default setting of the bit is the desired value (i.e. IA FW must write 0 to a bit even if the hardware reset value is 0). Doing so will ensure that the bit is unchangeable by non-System BIOS software.

Registers containing multiple R/WO bits must be programmed in a single atomic write. Failure to do so will result in non-programmable bits.



14.1.4 Power Optimizer Configuration

System BIOS must execute the following steps as part of System BIOS initialization of the APL SATA controller on both cold boot (G3/S5) and S3/S4 resume path.

Program the "SATA Initialization Register Index" (SIRI) register, SATA PCI offset A0h, to the appropriate index value, such as 64h

Program the "SATA Initialization Register Data" (SIRD) register, SATA PCI offset A4h, to the appropriate value, such as 9001h.

14.2 DevSleep Enabling

APL will be providing integrated HW support [AHCI] for the new DevSleep (Device Sleep) feature. While Hardware will provide autonomous support for DevSleep, some SW support is required (for example, initial discovery, initialization of the AHCI HBA and the attached storage device(s), interlock switch handling, and so on.).

During system POST, some additional activities need to be completed by BIOS during storage initialization. For each implemented port, BIOS shall perform the following actions on the AHCI HBA and associated storage devices:

1. Set CAP2.SDS (Supports DEVSLP) to '1'.
2. Set CAP2.SADM (Supports Aggressive DEVSLP Management) to '1'.
3. Set CAP2.DESO (DevSleep Entrance from Slumber Only) to '1'.
4. If the port provides inter-lock switch support or is hot pluggable or external, then skip the remaining steps for the port and repeat this step for the next available port. The rationale for this is:
 - On an interlock switch event, lack of SW awareness prevents SW from re-programming the PxDEVSLP registers as well as the storage device when a different device is inserted.
 - By spec, DevSleep and hot plug support are mutually exclusive.
 - By spec, DevSleep and external SATA support are mutually exclusive.
5. For each implemented port set PxDEVSLP.DSP (DEVSLP Present) to '1' if the port has DevSleep capabilities. Otherwise insure that this field is '0' (default).
6. If a storage device is connected to the port, issue IDENTIFY DEVICE command and using the storage device provided IDENTIFY DEVICE DATA, determine if the attached storage device supports DevSleep. If the attached storage device does not support DevSleep or , "DevSleep_to_ReducedPwrState", bit 7 of IDENTIFY DEVICE word 77 is '0b' then restart from Step #5 for the next available port.
7. If DevSlp support is not enabled; bit 8 of IDENTIFY DEVICE word 79 is '0b', enable the storage device to support DevSleep via SET FEATURES command (Count = 09h, Subcommand code = 10h).
8. Using the value specified in Identify Device Data log (log 30h page 08h QWord 6 (byte 48...55)[15:8]), program PxDEVSLP.DETO (DevSleep Exit Timeout). Set PxDEVSLP.DETO to 20ms (14h) if the Identify Device Data log (log 30h, 08h QWord 6 (byte 48...55)[15:8]) reports 0 or if the command fails.



Note: Make sure PxCMD.ST and PxDEVSLP.ADSE are cleared to '0' before updating PxDEVSLP.DETO value.

9. Using the value specified in Identify Device Data log (log 30h page 08h QWord 6 (byte 48...55)[4:0]), program PxDEVSLP.MDAT (Minimum DevSleep Assertion Time). Set PxDEVSLP.MDAT to 10ms (Ah) if the Identify Device Data log (log 30h, page 08h, QWord 6 (byte 48...55)[4:0]) reports 0 or if the command fails.

Note: Make sure PxCMD.ST and PxDEVSLP.ADSE are cleared to '0' before updating PxDEVSLP.MDAT value.

10. Set the PxDEVSLP.DM (DITO Multiplier) value to Fh. This allows OS SW to select the highest PxDEVSLP.DITO (DEVSLP Idle Timeout) value. The DevSleep Idle Timeout value is a port specific timeout value used by the AHCI for determining when to assert the DEVSLP signal. It provides a mechanism for the HBA to apply a programmable amount of hysteresis to prevent the HBA from asserting DEVSLP too quickly as this could result in undesirable latencies. While the platform BIOS will provide (program) a default value for this timer, there may be workloads where having a different (e.g., longer) idle timeout is desirable – it is the responsibility of OS SW to re-program the timeout value to the desired value. How the optimal value is determined by OS SW is beyond the scope of this document.

Note: Make sure PxCMD.ST and PxDEVSLP.ADSE are cleared to '0' before updating PxDEVSLP.DITO value.

11. Finally, program PxDEVSLP.ADSE (Aggressive DEVSLP Enable). This enables the AHCI to autonomously assert/de-assert DEVSLP.

Note: BIOS shall not program PxSCTL.IPM (Interface Power Management Transitions Allowed) with values other than 0-3h. Values of 4-7h are associated with DevSleep and will not be comprehended by non-DevSleep aware SW.

14.2.1 DevSleep Warm Reset Flow Handling

The GPIO pads that are configured to support DevSlp should have its PadRstCfg field set to '00b', allowing the SATA controller continues driving the deassert signal through warm reset path.

14.3 SATA Controller Disabling

Take the following steps to disable the SATA Controller.

If SATA is enabled by fuse and soft-strap but still need to be function disabled, the IA FW can disable SATA by steps below:

- Set AHCI BAR by programming SATA PCI register 0x24
- Enable command register memory space decoding
- Disable all ports by setting SATA PCI offset 94h[7:0] to all 0b
- Program AHCI MMIO VS_CAP register[12:1], PI register [5:0], and issue two reads to PI register
- Clear MSE and IOSE by SATA PCI offset 4h[1:0]



- Set SATA port clock disable bits by SATA PCI offset 90h[7:0] to all 1b
- Enable all SATA dynamic clock gating and dynamic power gating features by below steps:
 - a. Set SATA SIR offset 9Ch[29,23]
 - b. Set SATA SIR offset 8Ch[23:16, 7:0]
 - c. Set SATA SIR offset A0h[15]
 - d. Set SATA SIR offset 84h[23:16]
 - e. Program SATA SIR offset A4h[26, 25,24,14,5] = [0,0,0,1,1]
 - f. Set SATA SIR offset A8h[17:16]
- Disable SATA device by setting SATA PCI offset 9Ch [10]
- Set PCR [PSF1] AUD PCIEN[8] to 1
- Set AHCI BAR to zero

14.4 Known issues

SATA ODD Lost Detection Issue

Problem: When SATA ODD is connected to the SOC, the OS may experience media insertion or tray ejection detection failure.

Workaround: Intel Suggests customers to verify BIOS code with the following programming guidance.

SATA Initialization Register:

- Offset 0x80 – PTM1.PxSQOFFIDLED[x + 16] (x=0, 1, 2, 3, 4, 5, 6, and 7)
- Offset 0x90 – PTM5.PxPHYDPGE[x] (x=0, 1, 2, 3, 4, 5, 6 and 7)

In the reset of SATA Initialization steps:

- Clear SATA PCI CFG 0x98[24] = "0"
- Clear ABAR + Offset 0x24 [bit-3] = "0"
- Clear PxDEVSLP0.DSP = "0" (if SATA Port 0 is connected to an ODD).
- Clear PxDEVSLP1.DSP = "0" (if SATA Port 1 is connected to an ODD).

§



15 Intel® HD Audio

The Intel® High Definition Audio controller (Device #14, Function #0) is an internal PCI Express* endpoint device in APL. Software may access the Intel HD Audio controller registers (including the memory mapped registers) by byte, word, DWord quantities and on natural boundaries. Dword accesses must be on DWord boundaries, word accesses on word boundaries, and so on.

This chapter describes IA FW requirements for Intel® HD Audio controller support.

15.1 Intel® HD Audio Codec Initialization

This section involves the programming interface on Intel® HD Audio codec link. Readers are encouraged to read the relevant chapters of Intel® HD Audio Specification for information regarding architecture overview, register interface, programming model and codec features and requirements.

Intel® HD Audio allows flexible configurations of the inputs and outputs among its internal functional units and between codec and external jacks. Each pair of pins in the codec is assigned to an internal node in the codec, so the information related to the jack position, color coding, and so on. is mapped to the node that is internally assigned to the pins and wired to a jack. This information will allow the audio driver to configure the audio codecs correctly

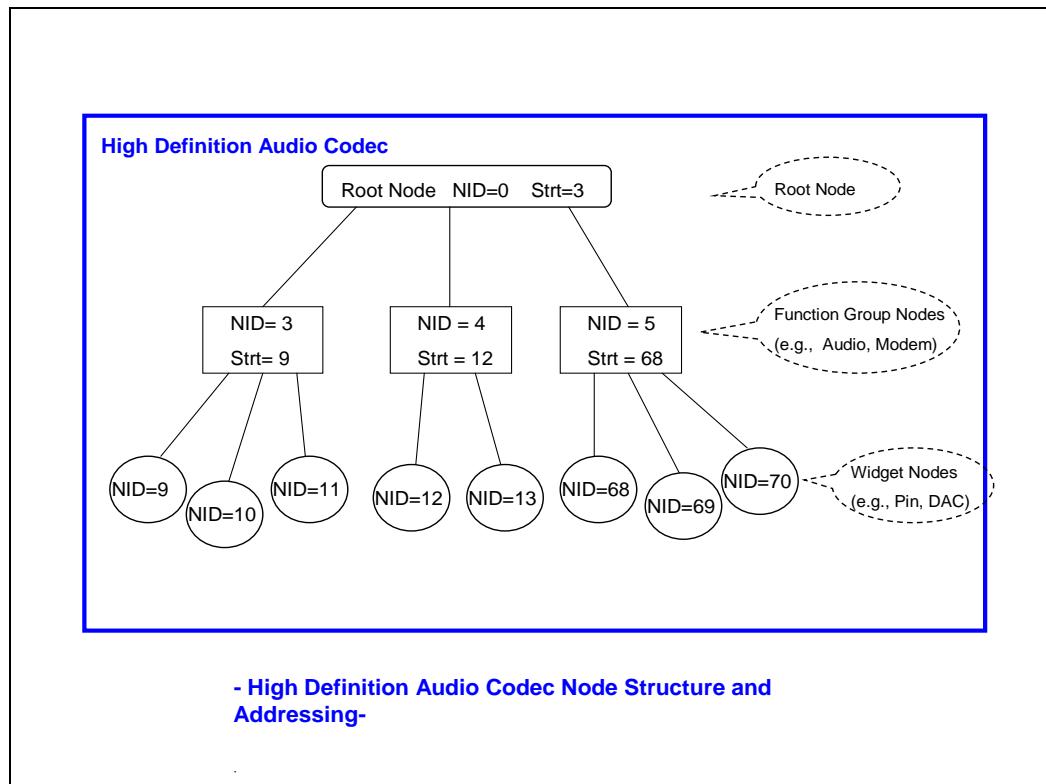
After IA FW has determined the presence of Intel HD Audio codec(s), it must follow the programming sequence given in this section to update the codec with correct jack information specific to the platform for the Intel HD Audio driver to retrieve and use later. If the codecs are not initialized with this platform-specific information, the Intel HD Audio driver will use the default data in the codecs which may or may not match the pin/jack connections or jack locations of the platform.

15.1.1 Intel® HD Audio Codec Architecture Introduction

The Intel HD Audio specification defines a modular codec architecture that is fully discoverable and configurable by software. It provides for the construction and description of various codec functions from a defined set of parameterized modules (building blocks or Widgets). Each such module and each collection of modules becomes a uniquely addressable Node, from which software can read capability parameters and to which it can send control commands. The root node is the top level node and always has a Node ID (NID) of 0. Each node contains information of the next level of nodes below it, in a tree structure as shown in [Figure 16-1](#).

For each Intel HD Audio codec present, a unique Codec Address (CAd) is assigned to the codec by hardware after reset during the codec link initialization and will be used for software to address each codec. For instance, the codec connected to SDIO (as indicated by HDBAR+0Eh[0]=1b) has its CAd=0, the codec connected to SDI1 (as indicated by HDBAR+0Eh[1]=1b) has its CAd=1, and so on. Each node in a codec has a pre-defined, unique Node ID (NID). The CAd+NID combination is used by a Verb to uniquely address a codec node.

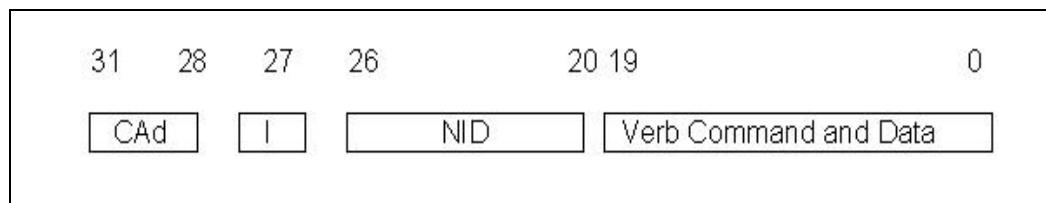
Figure 14-1. Intel® HD Audio Codec Node Structure and Addressing



A codec **verb** is a 32-bit DWord command sent to a codec by software, which contains the following information:

- Codec address and Node ID of the target node in the codec
- Command to be performed by the target node
- Data payload (if any)

Below is the format of a Verb DWord.



There are two ways for software to send verbs to and receive response data from codecs over the Intel HD Audio codec link: using CORB/RIRB (Command Output Ring Buffer / Response Input Ring Buffer) or using the Immediate Command/Immediate Response register pair. Refer the Intel® HD Audio Specification for details.



15.1.2 Codec Verb Table

For each codec present on the Intel HD Audio codec link, a corresponding pre-defined "Codec Verb Table" must be available to IA FW. The Codec Verb Tables are based on codec specific information (coded datasheet) and platform design specific information (schematics) and are built by IA FW writers and platform designers. The table contains a list of 32-bit "Verb"s (command and data payload) to be sent to the corresponding codec over the Intel HD Audio codec link.

Below is a sample Intel HD Audio Codec Verb Table for a platform with 1 codec at codec address 01h.

```
;Sample HIGH DEFINITION AUDIO Codec Verb Table
;Codec Address (CAd) = 01h
;Codec Vendor: XYZ Company
;VenID DevID:
    dd 12345678h
;-----
; FrontPanel_Supported? ; 1=Supported ,0=Not supported
    db 01h
; # of Rear Panel Pin Complexes
    dw 000Ch
; # of Front Panel Pin Complexes
dw 0002h
;-----

VerbTable0:

;Pin Complex 1      (NID 12h)
dd 01271F01h
dd 01271E01h
dd 01271D40h
dd 01271C10h
;Pin Complex 2      (NID 24h)
dd 02471F01h
dd 02471E01h
dd 02471D60h
dd 02471C11h
;Pin Complex 3      (NID 16h)
dd 01671F01h
dd 01671E01h
dd 01671D10h
dd 01671C12h
;Pin Complex 4      (NID 29h)
dd 02971F01h
dd 02971E01h
dd 02971D20h
dd 02971C14h
;Pin Complex 5      (NID 17h)
dd 01771F01h
dd 01771EA1h
dd 01771D90h
dd 01771C20h
;Pin Complex 6      (NID 15h)
dd 01571F01h
```



```

dd 01571E81h
dd 01571D30h
dd 01571C21h
;Pin Complex 7      (NID 18h)
dd 01871F99h
dd 01871E33h
dd 01871D11h
dd 01871C2Eh
;Pin Complex 8      (NID 1Ch)
dd 01C71F01h
dd 01C71EC4h
dd 01C71D11h
dd 01C71CF0h
;Pin Complex 9      (NID 1Bh)
dd 01B71F01h
dd 01B71E44h
dd 01B71D11h
dd 01B71CF0h
;Pin Complex 10     (NID 13h)
dd 01371F59h
dd 01371E1Fh
dd 01371D11h
dd 01371CF0h
;Pin Complex 11     (NID 1Ah)
dd 01A71F99h
dd 01A71E80h
dd 01A71D11h
dd 01A71CF0h

VerbTable0FP:
;Pin Complex 10     (NID 11h) Front Panel Jack
dd 01171F02h
dd 01171E21h
dd 01171D40h
dd 01171C30h
;Pin Complex 11     (NID 14h) Front Panel Jack
dd 01471F02h
dd 01471EA1h
dd 01471D90h
dd 01471C22h

```

15.1.3 Codec Initialization Programming Sequence

After IA FW has determined the presence of Intel HD Audio codecs, it must follow the programming sequence below to update the codec with the correct jack information specific to the platform for the Intel HD Audio driver to retrieve and use later.

There are two ways to send verbs to and receive response data from codecs over the Intel HD Audio codec link: using CORB/RIRB (Command Output Ring Buffer / Response Input Ring Buffer) or using the Immediate Command/Immediate Response register pair. The sequence below uses the latter which does not require the availability of a memory buffer.



IA FW should ensure that the Intel HD Audio HDBAR HDA PCI B0:D14:F0:10-17h contains a valid address value and is enabled by setting HDA PCI B0:D14:F0 :04h[1]. Before performing the codec initialization, IA FW should program HDA IOBCTL register 61Ch[9:8] via IOSF-SB port for I/O buffer ownership configuration. Detail as below:

- 00 → HDA-Link (HW default)
- 01 → HDA-Link and I2S port
- 11 → I2S port

IA FW must ensure program as mentioned in [section 14.5](#), The Controller Reset# bit of Global Control register in memory-mapped space (HDBAR+08h[0]) is set to 1b and read back as 1b. Additional delay might be required to allow codec coming out of reset prior to subsequent operations, contact your codec vendor for detail. When clearing this bit and setting it afterward, IA FW must ensure that minimum link timing requirements (minimum RESET# assertion time, and so on.) are met.

IA FW should do below steps:

- Set HDBAR + 08h[0] and read back as 1 for deassert HAD RESET# to start up the link.
- Read GCAP (HDBAR + 00h) and write same value back to the register, and then clear STATESTS bits for each of "SDIN Stat Change Status Flag" (HDBAR + 0Eh).
- Clear HDBAR + 08h[0] and read back as 0 to turn off the link.
- Set HDBAR + 08h[0] and read back as 1 to turn on the link.

For each Intel HD Audio codec present as indicated by HDBAR + 0Eh[3:0], IA FW should perform the codec initialization as described below:

- Read the VendorID/DeviceID pair from the attached codec.
- Read the Revision ID from the attached codec.
- Verify that the ICB bit, HDBAR + 68h[0], is 0.
- Write verb COOF0000h (DWord) to the IC register, HDBAR + 60h, where: 'c' (bits 31:28) represents the codec address (CAd).
- Program HDBAR + 68h[1:0] to 11b to send the verb to the codec.
- Poll the ICB bit, HDBAR+68h[0] until it returns 0 indicating the verb has been sent to the codec. IA FW may write HDBAR + 68h[0] to a 0 if the bit fails to return to 0 after a reasonable timeout period.
- If HDBAR + 68h[1] = 1b indicating the response data from the codec is now valid, read HDBAR + 64h; the data is the VID/DID value returned by the codec.
- Check against internal list to determine if there is a stored verb table which matches the CAd/VID/DID information.

Note: Steps 1 and 2 are IA FW implementation-specific steps and can be done in different ways. If an IA FW has prior knowledge of a fixed platform/codec combination (for example,, for a IA FW having 3 stored verb tables for 3 known codecs at known codec addresses on a known platform), a simple pre-defined codec-to-table matching can be used and steps 1 and 2 can be eliminated. For an IA FW to support multiple codec/platform combinations, an internal match-list might be needed to match a platform/codec combination to a codec verb table.



If there is a match, send the entire list of verbs in the matching verb table one by one to the codec.

- Verify the ICB bit, HDBAR + 68h[0] is 0.
- Write the next verb (DWord) in the table to HDBAR + 60h.
- Program HDBAR + 68h[1:0] to 11b to send the verb to codec.
- Poll the ICB bit, HDBAR + 68h[0] until it returns 0 indicating the verb has been sent to the codec. IA FW may write HDBAR + 68h[0] to a 0 if the bit fails to return to 0 after a reasonable timeout period.
- Repeat the steps until all the verbs in the table have been sent.

Note: Some verbs in the table may be dependent on certain platform-specific conditions. For example, for the sample table above, the verbs for Pin Complex 7 and 8 (NID=14,16 respectively) should be sent only if the Front Panel Jacks are present and connected on the platform, which may be indicated by a software flag that is controlled by a certain GPIO pin.

15.1.3.1 Codec Initialization Sample Code

Below is sample code of the Intel HD Audio codec initialization sequence.

```

;-----
;
; Procedure: Initialize High Definition AudioCodecs
;
; Description: Initialize High Definition Audio Codecs by sending
;               codec verbs to codecs.
;
; Input:
;       ES - 0000h with 4GB limit.
;       STACK - Available.
;       High Definition Audio controller's HDBAR is initialized and enabled.
;       Codec verb tables are available and defined in the
;       same code segment.
;
; Output:
;       CF : 1 = Codec initialization failure
;       CF : 0 = Codec initialization success
;
; Registers modified: All except segment registers.
;
; Notes:
;       MKF_AZALIA_BASE_ADDRESS = the value of HDBAR register
;       MKF_MAX_NUM_AZAL_CODECS = 3 (max of 3 codecs supported)
;       AZALIA_MMIO_STATESTS = 0Eh
;       AZALIA_MMIO_IC = 60h
;       AZALIA_MMIO_IR = 64h
;       AZALIA_MMIO_ICS_ICB = 68h
;       VerbHeaderSize = 11d
;-----
;

InitializeAzaliaCodecs PROC NEAR PUBLIC
;       ebx will always hold the High Definition Audio base address
;       mov     ebx, MKF_AZALIA_BASE_ADDRESS
;       ecx is the current codec address (only 15 codecs are supported in the
;               High Definition Audio spec so only the lower 4 bits are relevant)
;       mov     ecx, MKF_MAX_NUM_AZAL_CODECS

```



```
;      dx is the map of SDI pins, and the bits will be cleared as the
;      associated codecs are serviced
;      mov      dx, word ptr es:[ebx+AZALIA_MMIO_STATESTS]

InitCurrentCodec:
    dec      cx
    btr      dx, cx                      ; Test for 'cx'th codec
    jnc      NextSDI

;-----
;1.      Ensure High Definition Audio device is enabled and BARs are programmed
;
;   a. Program High Definition Audio BARs with temporary values
;   b. Enable memory space and bus mastering
;   c. Deassert CRST#
;-----
;       a. Set the AZ# bit to 1 to enable High Definition Audio signal mode
;(D27:F0:Reg40h[0]=1b)
        mov      ah, AZALIA_AZCTL_OFFSET
        mov      al, AZALIA_AZCTL_OFFSET_AZ
        _SET_PCI_FAR     AZALIA
;       b. Program the High Definition Audio HDBAR at PCI config space 10h-17h to
;a temporary address
        mov      ah, PCI_BAR0
        mov      ebx, MKF_AZALIA_BASE_ADDRESS
        _WRITE_PCI_DWORD_FAR   AZALIA
;       c. Enable memory space and bus mastering for High Definition Audio
        mov      al, CMD_MEM_SPACE+CMD_BUS_MASTER
        _SET_PCI_FAR     Azalia
;       d. Deassert the CRST bit in High Definition Audio to cause the link to
;start up(HDBAR+08h[0]=1)
        or       byte ptr es:[ebx+AZALIA_MMIO_GCTL], AZALIA_MMIO_GCTL_CRST

;-----
;2.      Read the Vendor ID/Device ID pair from the attached codec
;
;   a. Poll the ICB bit in the ICS register at HDBAR+68h[0] until it returns 0
;   b. Write verb c00F0000h (dword) to the IC register at HDBAR+60h; where 'c'
;       (bits 31:28) represents the codec address (CAd).
;   c. Set bits 1:0 of the IRS register at HDBAR+68h[1:0]
;   d. Poll ICS register bits at HDBAR+68h[1:0] until they return 10b indicating
;the verb has been sent to the codec and response data from codec is now valid.
;   e. Read IR register at HDBAR+64h, the dword data is the VendorID/Device
;       ID value returned by the codec
;-----
;   a. Poll the ICB bit in the ICS register at HDBAR+68h[0] until it returns 0

        push      cx
        xor      cx, cx                      ; 64K cycles
PollICBBit:
        test      word ptr es:[ebx+AZALIA_MMIO_ICS], AZALIA_MMIO_ICS_ICB
        jz       ICBBitClear                ; Poll ICB bit until it returns 0
        loop      PollICBBit
;       Add error handling code here
;       When we timeout, reset link per audio driver team request
        and       byte ptr es:[ebx+AZALIA_MMIO_GCTL], NOT AZALIA_MMIO_GCTL_CRST
        or        byte ptr es:[ebx+AZALIA_MMIO_GCTL], AZALIA_MMIO_GCTL_CRST
ICBBitClear:
        pop      cx

;   b. Write verb c00F0000h (dword) to the IC register at HDBAR+60h; where 'c'
;       (bits 31:28) represents the codec address (CAd).
```



```

        mov      eax, ecx
        shl      eax, 28
        or       eax, 000F0000h
        mov      dword ptr es:[ebx+AZALIA_MMIO_IC], eax ; Write the verb
;      c. Set bits 1:0 of the IRS register at HDBAR+60h[1:0]
        or       word ptr es:[ebx+AZALIA_MMIO_ICS], BIT1+BIT0 ; Send the command

;      d. Poll ICS register bits at HDBAR+68h[1:0] until they return 10b indicating
; the verb has been sent to the codec and response data from codec is now valid.
PollDataValid:
        mov      al, byte ptr es:[ebx+AZALIA_MMIO_ICS]
        cmp      al, 10b
        jne      PollDataValid

;      e. Read IR register at HDBAR+64h, the dword data is the VendorID/Device
; ID value returned by the codec
        mov      eax, dword ptr es:[ebx+AZALIA_MMIO_IR] ; eax=vendorID/deviceID

;-----
;3.   Check against the list of supported vendor ID/Device ID combinations
; to determine if the received VID/DID is supported.
;-----
        push     ecx

        call     CheckforValidCodec
        or       cx, cx
        jz      VerbTableDone           ; jump if VID/DID not supported

;-----
;4.   If there is a match, send the entire list of verbs in the matching verb
; table one by one to the codec
;      a. Poll the ICB bit of the ICS register at HDBAR+68h[0] until it returns 0.
;      b. Write the next verb (dword) in the table to the IC register at HDBAR+60h.
;      c. Write the bits of the ICS register at HDBAR+68h[1:0] to 11b to send the
; verb to the codec.
;      d. Repeat steps 4a-4c until all verbs in the table have been sent for the
; current codec.
;-----
;      a. Poll the ICB bit of the ICS register at HDBAR+68h[0] until it returns 0.

        push     cx
        xor      cx, cx
PollicBBit2:
        test    word ptr es:[ebx+AZALIA_MMIO_ICS], AZALIA_MMIO_ICS_ICB
        jz      ICBBit2 ; Poll ICB bit until it returns 0
                  ;(need to change "Azalia" in this command to HDAudio?
        loop    PollicBBit2
;      Add error handling code here
ICBBit2:
        pop      cx

;b. Write the current verb (dword) in the table to the IC register at HDBAR+60h.
        mov      eax, dword ptr cs:[si]
        mov      dword ptr es:[ebx+AZALIA_MMIO_IC], eax ; Write verb

;      c. Write the bits of the ICS register at HDBAR+68h[1:0] to 11b to send the
; verb to the codec.
        or       word ptr es:[ebx+AZALIA_MMIO_ICS], BIT1+BIT0
;      d. Repeat steps 3a-3d until all verbs in the table have been sent for the
; current codec.

        loop    PollicBBit2           ; Continue until all verbs written

```



```
VerbTableDone:  
    pop      ecx  
  
NextSDI:  
    or       dx, dx  
    jnz     InitCurrentCodec  
  
AzaliaCodecComplete:  
    ret  
InitializeAzaliaCodecs  ENDP  
  
-----  
;  
; Procedure:  CheckforValidCodec  
;  
; Description:Detects whether the vendor and device ID of the current codec  
; is supported based on whether the value is found at the start  
; of any of the codec verb tables.  
;  
; Input:   EAX      - Vendor and device ID of the current codec  
;          ECX      - Current codec address  
;          DS       - BDA_DSEG.  
;          ES       - 0000h with 4GB limit.  
;          FS       - POST_DSEG.  
;          GS       - RUN_CSEG.  
;          STACK    - Available.  
;  
; Output: CX      - Size of codec verb table (in dwords) if a valid  
;                  codec is present. Else cx = 0.  
;          SI       - Address of the codec verb table (valid if CF=0)  
;  
; Modified: SI  
-----  
  
CheckforValidCodec      PROC NEAR PUBLIC  
    push    bx  
    push    edx  
    push    si  
  
    xor    bx, bx  
CheckNextCodecTable:  
    mov    si, word ptr cs:[bx+offset CodecVerbTableList]  
    cmp    dword ptr cs:[si], eax  
    je     FoundValidCodec  
  
;      end of table?  
    add    bx, 2                      ; Next verb table entry  
    cmp    bx, (offset CodecVerbTableListEnd - offset CodecVerbTableList)  
    jb     CheckNextCodecTable  
  
CodecNotValid:  
    xor    cx, cx  
    jmp    CodecCheckDone  
  
FoundValidCodec:  
    mov    edx, dword ptr cs:[si+VerbHeaderSize]           ; Get first verb  
    shr    edx, 28  
    cmp    edx, ecx  
    jne    CodecNotValid  
    add    si, 6  
    call   GetVerbTableSize  
          ; Codec has valid DID/VID and addr
```



```

CodecCheckDone:
    pop     si
    pop     edx
    pop     bx
    ret

CheckforValidCodec      ENDP

;-----
;

; Procedure:   GetVerbTableSize
;
; Description: Checks the front panel sensing GPIO to determine if front
;               panel jacks are present. The routine returns the size of
;               the verb table (size may depend on whether front panel is
;               supported or if the codec supports front panel).
;
; Input: SI      - Address of the front panel supported status byte
;        DS     - BDA_DSEG.
;        ES      0000h with 4GB limit.
;        FS     - POST_DSEG.
;        GS     - RUN_CSEG.
;        STACK   - Available.
;
; Output: CX      - Size of codec verb table in dwords
;          SI      - Address of the codec verb table
;
; Modified: EBX, CX, SI
;-----


GetverbTableSize      PROC NEAR PUBLIC
    push    ebx

    mov     cl, byte ptr cs:[si]           ; al = Front panel support bit
    inc     si

    or      cl, cl
    mov     cx, word ptr cs:[si]          ; cx = length of rear panel table
    jz     FPSupportDone                ; If front panel not supported
                                         ; by the codec, no need to add
                                         ; FP table size

; TODO: OEMs must add code here to query the GPIO dedicated to front
;       panel sensing.
    jz     FPSupportDone
; If control comes here, front panel jack is supported by the codec and
; is present in the system, so add the size of the FP table.
    add     cx, word ptr cs:[si+2]         ; cx = rear panel table size +
                                         ; front panel table size

FPPSupportDone:
    add     si, 4                         ; si = start of codec verb table
    shl     cx, 2                         ; cx = # of Pin complexes * 4
                                         ;      = # of dwords in table
    pop     ebx
    ret

GetVerbTableSize      ENDP

CodecVerbTableList:
    dw offset VerbTable0
    dw offset VerbTable1

```



CodecVerbTableListEnd:

15.1.4 Intel® HD Audio Codec Initialization on S3 Resume

According to Microsoft*, the SSID response from the Intel HD Audio codec must be consistent; at any point the OS may read the value. Similarly other codec configuration information must follow the same rule. Additionally, the assumption of relying on the function driver to ensure such consistency across different sleep states is not always practical due to the fact that the function driver can be disabled/unloaded by the user prior to system power state transitions.

This requires that programming of the Intel HD Audio codecs performed by the IA FW during POST must also be performed any time the codec loses power. In particular, this means that the codec verb table, if programmed into the codec during POST, must be restored by IA FW on an S3 resume if the codec context is not constantly maintained by standby power.

Note: This requirement does not apply to platforms that use the codec's hardware default configurations without changing them, or platforms that maintain the codec context in S3 by standby power.

The IA FW programming on resume must preserve the wake status information in the codec, so that the driver can determine if a codec (usually a modem) has 'awoken' the system. The following programming sequence is therefore recommended during S3 resume:

1. Program as mentioned in [section 15.4](#).
2. Set HDBAR + 08h[0] to take the controller out of reset, wait for about 1ms.
3. Program Verb Tables to codecs as IA FW did during the POST.

15.2 Intel® HD Audio Controller Configuration

Once the Intel® High Definition Audio codec is determined to be present and the Intel® High Definition Audio controller is, IA FW should:

- Initialize the configuration space of Intel HD Audio controller as a regular PCI device (assign memory, interrupt resources, and enable the device using standard PCI command register 04h).
- Initialize SID/SVID registers at HDA PCI B0:D14:F0:2Ch~2Eh to OEM-specific IDs. This is similar to the SID/SVIDs given to other devices such as SATA, SMBus, USBs and so on.

15.3 Intel® HD Audio PME Event

The PME signal is used to let AUC exit from D3 state on the event of audio or HDMI jack insertion. IA FW could config HDA PME enable in HDA BAR + 0x0C.

To support this PME feature in an ACPI OS environment, IA FW needs to provide the proper _PRW object and \GPE._L6D() control method in the ACPI name space.



15.4 Intel / iDisplay Audio Link

The Intel / iDisplay Audio link is used to feed audio data streams from AUC to display controller, which will embed audio into the combined audio/video data stream sent through DisplayPort or HDMI output interfaces.

APL codec can be addressed by using CAd==2.

15.5 Additional Intel® HD Audio Programming Steps

15.5.1 PCI Mode

HDA should be operating under PCI mode, IA FW should set PCR[HDA] + 530h[1]

It is expected only the PCI mode is supported in such the ACPI mode is to be disabled. The default has been PCI mode and thus nothing to be done at the IP level.

15.5.2 Function Disabling Steps

1. Put device in D0i3 by setting HDABA + 104Ah[2]
2. Put device in D3hot by setting PCI[HDA] + 54h[1,0]
3. Clear FNCFG register PCR[HDA] + 530h [5], and set PCR[HDA] + 530h [4][2][0]
4. Disable PCI function by setting PCR[PSF3] + 0x201C[8]
5. Function disable at PBASE+0x34[28]

15.5.3 Miscellaneous Initialization Steps

1. Program PC.PMES, HDA PCI offset 52h[15:11] = 18h before locking FNCFG.BCLD.
2. The Audio PLL configuration setting is not at the optimal value by default. IA FW is required to program with the expected setting before controller is out of reset (Refer 15.5.4).
 - a. APLL0.KPKISC, PCR[HDA] + 610h [31:30] = 11b
 - b. APLL0.KIS, PCR[HDA] + 610h [29:28] = 11b
 - c. APLL0.KPS, PCR[HDA] + 610h [27:26] = 11b
 - d. APLL0.PLLWC, PCR[HDA] + 610h [22:16] = 1Eh
 - e. APLLP1.DCOTC, PCR[HDA] + 614h [13:8] = 3Fh
 - f. APLLP2.KPCE, PCR[HDA] + 618h [11:8] = 1h
 - g. APLLP2.KICE, PCR[HDA] + 618h [4:0] = 1Dh
3. Per IA FW policy, program the “Ownership Selection”, PCR[HDA] + 61Ch[9:8], according to platform configuration (Azalia vs I2S).

15.5.4 Static Frequency Switching

IA FW is required to perform the additional steps listed below to statically switch BCLK clock frequency.

1. If iGFX is to be disabled,



2. set iDisplay Link's LCTLx.SPA to 0
3. skip all other iDisplay Link related steps
4. For iDisplay Link, platform will determine is T-mode if to be set to 1T or 2T. (Legal Freq-T-mode pairings are 96MHz-2T, 48 MHz, either T).
5. Set the following registers before iDisplay Link is brought out from reset (LCTLx.SPA = 0b, poll LCTLx.CPA = 0b):
 6. Set SEM1.TMODE, HDA PCI offset C0h [12] = 0b for 2T or 1b for 1T Mode
 7. Set T mode in the display codec using iGFX AUD_FREQ_CNTRL MMIO register access:
 8. GttMmAddr + 0x65900[15] = 0b for 2T or 1b for 1T Mode
 9. Set iDisplay Link freq using iGFX AUD_FREQ_CNTRL MMIO register access, GttMmAddr + 0x65900[4:3]:
 10. 96 MHz: 10b
 11. 48 MHz: 01b
 12. For HDAudio Link, set Output and Input Payload Capability using HDA MMIO registers access after controller out of reset (set GCTL.CRSTB, HDABAR + 8h bit[0] = 1b):
 13. OUTPAY: HDABAR + 0x04[15:0]
 14. INPAY: HDABAR + 0x06[15:0]
 15. accordingly:
 16. 24 MHz:
 17. OutputPayloadWords = 0x3C (60 words of payload)
 18. InputPayloadWords = 0x1D (29 words of payload)
 19. 12 MHz:
 20. OutputPayloadWords = 0x1C (28 words of payload)
 21. InputPayloadWords = 0x0D (13 words of payload)
 22. 6 MHz:
 23. OutputPayloadWords = 0x0D (13 words of payload)
 24. InputPayloadWords = 0x05 (5 words of payload)
 25. For each link:
 26. Platform will determine the optimal supported link frequency and passed in as BIOS policy
 27. Check if connected codec supports required BCLK frequency by sending codec command with Verb ID = F00h, Parameter ID = 16h
 28. Turn off the link: set LCTLx.SPA to 0, poll LCTLx.CPA till it is 0
 29. Set LCTLx.SCF accordingly
 30. Turn on the link: set LCTLx.SPA to 1, poll LCTLx.CPA till it is 1
 31. Wait 512us
 32. For non-S3 path, for each link, get current frequency from all codec by sending codec command with Verb ID F37h: GET_CCF
 33. If any of the results does not match the desired frequency, perform the below step to revert back to its default link frequency:
 34. Set LCTLx.SPA to 0
 35. Poll LCTLx.CPA till it is 0
 36. Set LCTLx.SCF back to default frequency



37. Set LCTLx.SPA to 1

15.5.5 Registers Lock Down and Related Field

Set PCR[HDA] + 30h[4] = 1b after all settings done to lock down register. Since there are other register fields sharing with this bit, their related programming have to be done at the same time.

Note: This step cannot be done before the interrupt assignment steps which write to the HDA's INTx register.

§



16 Intel® Smart Sound Technology (Intel® SST)

The Intel® Smart Sound Technology (Intel® SST) Controller in APL is a capability block within the Audio device.

This section describes IA FW requirements for Intel® SST controller support. For anything that is common with the Audio, refer [Section 15](#) for APL HDA.

16.1 Intel® Smart Sound Technology Initialization Flow

It is expected only the PCI mode is supported in such the ACPI mode is to be disabled. The default has been PCI mode and thus nothing to be done at the IP level.

16.2 Intel® Smart Sound Technology Enable/Disable

There could be SKU where the Intel® SST is fused disable. It can be determined by reading the PCR[HDA] + F0h [1] being 1.

For SKU that supports Intel® SST, the BIOS can choose to disable it by policy. Refer [Section 15.5.2](#)

To enable Intel® SST, the BIOS should:

1. Set the following policies:

```
ScPolicy->HdAudioConfig.DspEnable = TRUE;  
ScPolicy->HdAudioConfig.DspUaaCompliance = FALSE;
```

This applies the programming: set Sub Class Code to 01 for Audio DSP enabled, HDA PCI offset 0Ah [7:0] to 1, to allow Intel® SST driver loading.

2. Set "GPROCEN", HDBAR + 804h [30] to 1

For Intel® SST enabled system that is required to support also Audio driver,

BIOS shall set the following policy:

```
ScPolicy->HdAudioConfig.DspUaaCompliance = TRUE;
```

The policy applies the following programming:

3. Set Sub Class Code to 03 (default), HDA PCI offset 0Ah [7:0] to 3.\
4. Set Programming Interface to 01, HDA PCI offset 09h [7:0] to 1, to allow both Intel® SST and Inbox HDA driver loading.

The above is needed only on systems not supported by Intel® SST driver.

(Like Windows* 7 and older) and results in DSP functions are not available.



16.3 ACPI Requirements

The offload engine driver invokes the _DSM method that is used for obtaining information such as the Non-HD-A Link Descriptor Table or checking if a pre- or post-processing module is supported.

```

// Arg0 — UUID: A69F886E-6CEB-4594-A41F-7B5DCE24C553 (Buffer)
// Arg1 — Revision ID: 0x01 (Integer)
// Arg2 — Function Index: 0x0 - 0x4 (Integer) - See below for details.
// Arg3 — Depends on Function Index - See below for details.
// Return - Depends on Function Index - See below for details.
Method (_DSM, 4, Serialized, 0, UnknownObj, {BuffObj, IntObj, IntObj,
PkgObj}) {

    // Verify UUID
    If (LEqual(Arg0, ToUUID ("A69F886E-6CEB-4594-A41F-7B5DCE24C553")){

        Switch (Arg2) {

            // Function 0: Function Support Query
            // Arg2 - Function Index: 0x00 (Integer)
            // Arg3: Unused
            // Return: Bitmask of functions supported. (Buffer)
            Case (0) {
                // Supports function 0-5
                Return(Buffer(One) { 0x3F })
            }

            ...

            // Function 5: aDSP Resource Configuration Settings
            //             Retrieve configuration settings indicating how to
            reserve
                //             aDSP memory, compute, and DMA resources.
                // Arg2 — Function Index: 0x05 (Integer)
                // Arg3 — Unused
                // Return: Buffer containing resource configuration
                settingvalues. See SwAS for
                    //             more details.
                Case (5) {
                    Return(RBUF)
                }

                Default {
                    Return (0)
                }
            }
        }
    }
}

```

16.3.1 Non-HD-A Link Descriptor Table

ACPI Table for non-HD Audio devices (Digital Microphones and BT Sideband over I2S*).

1. Enable/Disable option in BIOS
2. Link Configurations
3. Microphone Description (Location/Array Geometry)

The offload engine driver retrieves this ACPI table by executing the following _DSM method:

Arg0 — UUID: A69F886E-6CEB-4594-A41F-7B5DCE24C553 (Buffer)

Arg1 — Revision ID: 0x01 (Integer)

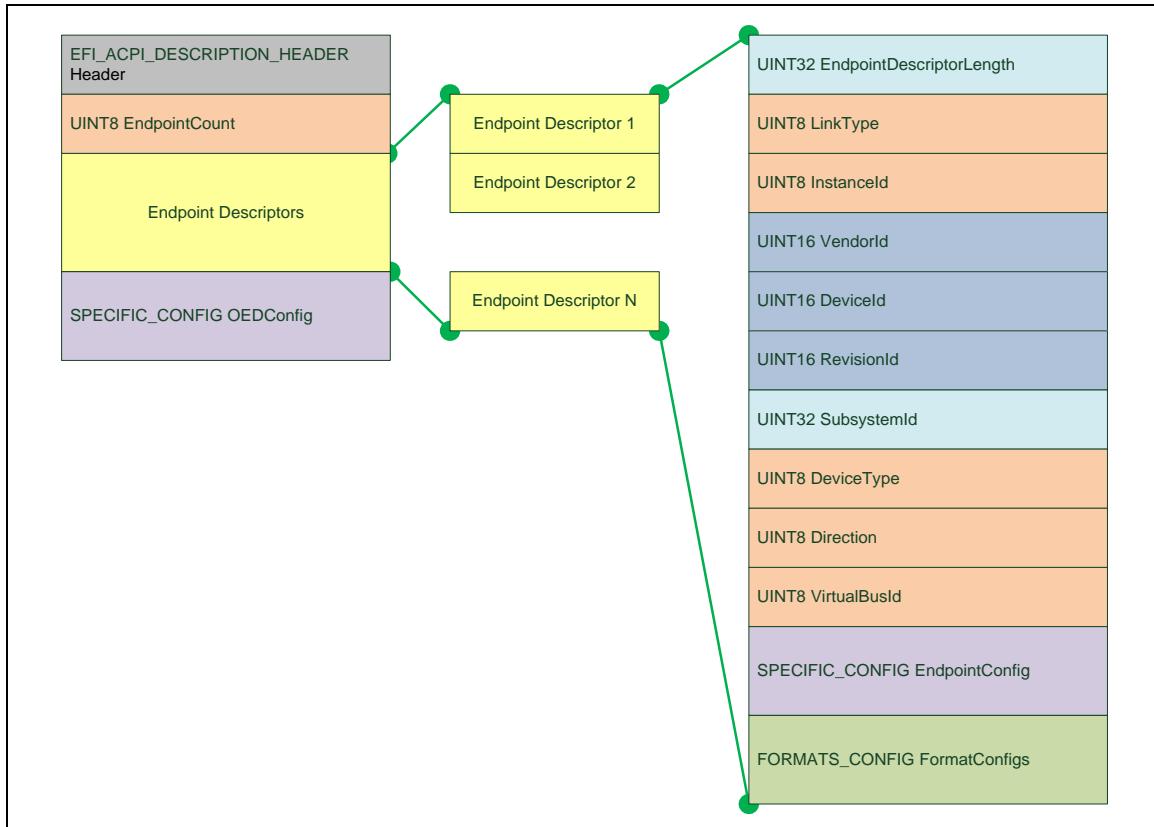
Arg2 — Function Index: 0x01 (Integer)

Arg3 — Depends on Function Index

Return — ACPI Table describing the non HD Audio links and devices supported by the audio DSP. (Buffer)

ACPI configuration table is defined as follows:

Figure 15-1. ACPI Configuration Table



Where used data types are defined as follows:

```
typedef struct _SPECIFIC_CONFIG
{
```



```

        UINT32      CapabilitiesSize;
        BYTE[]       Capabilities;
    } SPECIFIC_CONFIG;

typedef struct _FORMAT_CONFIG
{
    WAVEFORMATEXTENSIBLE      Format;
    SPECIFIC_CONFIG           FormatConfiguration;
} FORMAT_CONFIG;

typedef struct _FORMATS_CONFIG
{
    UINT8                    FormatsCount;
    FORMAT_CONFIG[]          FormatsConfiguration;
} FORMATS_CONFIG;

```

Where standard datatypes are defined as follows (simplified format):

```

typedef struct {
    WORD   wFormatTag;
    WORD   nChannels;
    DWORD  nSamplesPerSec;
    DWORD  nAvgBytesPerSec;
    WORD   nBlockAlign;
    WORD   wBitsPerSample;
    WORD   cbSize;
    WORD   wValidBitsPerSample;
    DWORD  dwChannelMask;
    GUID   SubFormat;
} WAVEFORMATEXTENSIBLE;

typedef struct {
    UINT32  Signature;

```



```
UINT32 Length;
UINT8 Revision;
UINT8 Checksum;
UINT8 OemId[6];
UINT64 OemTableId;
UINT32 OemRevision;
UINT32 CreatorId;
UINT32 CreatorRevision;
} EFI_ACPI_DESCRIPTION_HEADER;
```

Note: All the mentioned structures are 1-byte packed.

16.3.2 Feature Support

The offload engine driver retrieves this feature by executing the following _DSM method:

- Arg0 — UUID: A69F886E-6CEB-4594-A41F-7B5DCE24C553 (Buffer)
- Arg1 — Revision ID: 0x01 (Integer)
- Arg2 — Function Index: 0x02 (Integer)
- Arg3 — Unused
- Return — bitmask of supported feature (1 means enabled)

Bit	Description
0	WoV
1	BT sideband
2	codec based VAD
3-4	RSVD
5	BT Intel HFP SCO
6	BT Intel A2DP
7-31	RSVD

Bitmask of supported DSP Features is set through PCH Policy: "DspFeatureMask" Bitmask with the above structure (For example for WoV: ScPolicy->HdAudioConfig.DspFeatureMask |= BIT0).

16.3.3 3rd Party IP

3rd Party Processing Module supports bitmask, this is set through Policy: "DspPpModuleMask" Bitmask and has the following values:



Bit#	3rd Party IP
0	Waves
1	DTS
2	Spatial
3	Dolby
4-6	RSVD
7	ForteMedia
8	Intel WoW
9	Sensory WoW

Bitmask must be consistent with HD Audio ACPI _DSM Function 3 implementation.

16.3.3.1 ACPI _DSM Function 3

For 3rd party IP such as Waves and DTS, similar method will be used to enable/disable 3rd party Post or Preprocessing IP, with further dedicated GUID as defined by the 3rd party IP module suppliers:

Arg0 – UUID: A69F886E-6CEB-4594-A41F-7B5DCE24C553 (Buffer)
 Arg1 – Revision ID: 0x01 (Integer)
 Arg2 – Function Index: 0x03 (Integer)
 Arg3 – UUID: Specifies the UUID of the Post Processing module to check (Buffer)
 Return – TRUE if PP module is supported, else FALSE

The Example 2 Bitmask is set to enable IP “Dolby” support in Sc Policy initialization:

HdaConfig->DspPpModuleMask |= BIT3; // Enable Dolby support and _DSM is implemented to check 3rd party IP GUID. If it is matched, it will return this IP's bit value in DspPpModuleMask. In this example, GUID is a dummy value, OEM should place the valid GUID received from 3rd party IP provider.

Example 16-1. _DSM Function 3

```

Device(HDAS) {
  [...]
  Method(_DSM, 0x4, NotSerialized, 0, UnknownObj, {BuffObj, IntObj,
  IntObj, PkgObj}) {
    // Arg0 - UUID: A69F886E-6CEB-4594-A41F-7B5DCE24C553 (Buffer)
    // Arg1 - Revision ID: 0x01 (Integer)
    // Arg2 - Function Index: 0x0 - 0x3 (Integer) - See below for
    // details.
    // Arg3 - Depends on Function Index - See below for details.
    // Return - Depends on Function Index - See below for details.

    // Verify UUID
  }
}

```



```
If (LEqual(Arg0, ToUUID ("A69F886E-6CEB-4594-A41F-7B5DCE24C553")))
{
[...]
    // Function 3: Query 3rd Party Processing Module Support
    // Used by the Intel Offload Engine Driver to determine if
a
    // specified 3rd Party Module is allowed to be supported on
    // this platform
    // Arg2 - Function Index: 0x03 (Integer)
    // Arg3 - UUID: Specifies the UUID of the PP module to
check
    // (Buffer)
    // Return - TRUE if PP Module supported, else FALSE.
Case (3) {
    // ADPM - NVS AudioDSP Post-Processing Module Bit Mask
    // updated from PchPolicy: HdaConfig->DspPpModuleMask

    // Dolby support (enabled by policy
    // HdaConfig->DspPpModuleMask |= BIT3)
    // TO BE UPDATED WITH REAL GUID FOR DOLBY
    If (LEqual(Arg3, ToUUID ("AABBCCDD-EFFF-1122-3344-
556677889900"))){
        Return(And(ADPM, BIT3)) // DspPpModuleMask[BIT3] /
                                // ADPM[BIT3] set - Dolby
                                // supported (return true)
    }
    //
    // Implement for all supported PP modules
    //
}
}}
```

§



17 System Management Bus

APL provides a System Management Bus (SMBus) host controller as well as a SMBus Slave Interface. The host controller provides a mechanism for the processor to initiate communications with SMBus peripherals (slaves).

17.1 SMBus Usage Model

The Host Busy bit located in the SMBus Host Status Register, $\text{SMBBBASE} + 00h[0]$, can be cleared before the correct data is loaded in the data register. The wrong data may be read from the data of the SMBus controller. The IA FW should use the INTR bit, $\text{SMBBBASE} + 00h[1]$, and the other error indicator bits in the SMBus Host Status Register of SMBus I/O Space to indicate the end of the operation before reading the SMBus data register.

Software should ensure that all the SMBus transfers are completed before executing the I/O port CF9h reset. The Host Busy bit, in the SMBus Host Status Register, $\text{SMBBBASE} + 00h[0]$, should be zero indicating that the SMBus Host controller is not busy. Also if the Host Busy bit does not go to a zero for a specified amount of time (about 15ms), a SMBus kill command should be executed to terminate the transfer before a CF9h-based system reset can be executed on the system.

After the execution of the SMBus Kill Command a delay of at least 15ms should be provided before the CF9h reset.

17.2 Reading a Byte of SMBus EEPROM Data

The following steps have to be followed for the System BIOS to read a byte of the Serial Presence Detect (SPD) data from the DIMM.

1. Program the SMBus Base Address Register, PCI offset 20h.
2. Set the I/O Space enable bit, PCI offset 04h[0].
3. Set the HST_EN bit, PCI offset 40h[0].
4. Clear the status bits by programming $\text{SMBBBASE} + 00h$ to 1Eh.
5. Program the SMBus Transmit Slave Address Register with the DIMM SMBus address, $\text{SMBBBASE} + 04h$. For example: If the DIMM address is A2h and a byte is to be read from the DIMM, program $\text{SMBBBASE} + 04h$ to A3h (A2h OR'ed with 1 (Read/Write bit)).
6. Program the SMBus Host Command Register with the DIMM's SPD data offset to be read, $\text{SMBBBASE} + 03h$. For example: If reading from offset 04h in the DIMM's SPD, this register will have a value of 04h (Offset of SPD data to be read).
7. Program the SMBus Host Control Register, $\text{SMBBBASE} + 02h[7:0]$ to 48h.
8. Wait on the Host Busy (HOST_BUSY) bit to be cleared in the Host Status Register, $\text{SMBBBASE} + 00h[0]$. The System BIOS should use the INTR bit, $\text{SMBBBASE} + 00h[1]$ and the other error indicator bits in the SMBus Host Status Register to indicate the end of the operation before reading the SMBus data register.
9. After the HOST_BUSY bit is cleared, check the DEV_ERR bit in the Host Status Register, $\text{SMBBBASE} + 00h[2] = 0$, if the HOST_BUSY bit is cleared and the



DEV_ERR bit is cleared the IA FW can read the byte value from the SMBus Data 0 Register, SMBBASE + 05h.

Step #1 to Step #3 will be programmed only once. The BIOS can follow Step #4 through Step#9 multiple times to get all the necessary EEPROM data from the SMBus devices.

17.3 Block Mode

The block mode is not automatic and the software must perform the byte transfers. This is done differently for reads and writes.

17.3.1 Block Writes

Block writes work in the following manner:

1. Obtain ownership of the SMBus by using the In Use Status (INUSE_STS) bit, SMBBASE + 00h[6].
2. Clear the Byte Done Status (DS) bit, SMBBASE + 00h[7].
3. Set the INTREN, SMBBASE + 02h[0], if using interrupts.
4. Set the Command Byte in the HST_CMD Register, SMBBASE + 03h.
5. Set the Transmit Slave Address Register, SMBBASE + 04h, to the address of the device to be written, with bit 0 cleared to indicate a write.
6. Load the count into the Data 0 (HST_D0) register, SMBBASE + 05h.
7. Write the data byte into the Host Block Data Register, SMBBASE + 07h.
8. Issue the block write command.
9. Either poll Byte Done Status, SMBBASE + 00h[7], or wait for the interrupt, if enabled.
10. If this is the last byte, then clear the byte done status bit and the INUSE_STS bit and end. Otherwise continue with step 11.
11. Write the next byte into the Host Block Data Register, SMBBASE + 07h.
12. Clear the byte done status bit and repeat step 9.

17.3.2 Block Reads

Block reads work in the following manner:

1. Obtain ownership of the SMBus by using the In Use Status (INUSE_STS) bit, SMBBASE + 00h[6].
2. Clear the Byte Done Status (DS) bit, SMB_BASE + 00h[7].
3. Set the INTREN bit, SMB_BASE + 02h[0], if using interrupts.
4. Set the Transmit Slave Address Register, SMB_BASE + 04h, to the address of the device to be read, with bit 0 set to indicate a read.
5. Issue the block read command
6. The first byte transferred is the Count and does not generate an interrupt or set the status bit.
7. Either poll the DS bit, SMB_BASE + 00h[7], or wait for the interrupt, if enabled.
8. Read the byte from the Host Block Data Register, SMB_BASE + 07h.



9. If the number of bytes received = Count – 1, set the LAST_BYTE bit in the Host Control Register, SMBBASE + 02h[5].
10. Clear the DS bit. If the last byte then end, otherwise loop to step 7.

17.4

BIOS / Driver Hardware Semaphore

The In Use Status bit, to enable a driver to share the SMBus with the System BIOS. System BIOS must use this bit when it needs to access any SMBus device after the OS loaded. To use this bit, read the Host Status Register, SMB_BASE + 00h[6] until it reads a 0. The act of reading this bit will set it to 1 for all subsequent reads. Once this bit has been read as a 0, System BIOS is free to use the SMBus. After IA FW finished using the SMBus, System BIOS must write a 1 to the INUSE_STS bit in order to release the semaphore. This enables the SMBus to be used by another requestor.

If the System BIOS reads a 1, then the transaction should be retried some time later, probably scheduled by using the SMI delay timer (64ms).

17.5

Wake Up from SMBus

The System BIOS should set SMB_SMI_EN bit, PCI offset 40h[1], to enable the generation of a SMI from the SMBALERT# signal and the SMBus.

17.6

SMBus Interaction with TCO

System BIOS must not use the SMBus controller for 25 ms after setting the "send now" TCO bit. At boot time, a TCO message may be transmitted due to a watchdog timeout if enabled. In this case, the SMBus controller may not finish a command issued by the System BIOS for as much as 50ms. SMBus timeouts implemented by the System BIOS should always be 50 ms or longer.

System BIOS should ensure that the Host Status register of the SMBus is in a properly configured state prior to OS handoff. Leaving bits [5:1] set in the SMBus Host status register can cause the interrupt service routine of an OS device driver developed for a different device, to be continually executed by the OS. This behavior will occur in cases where devices are sharing an interrupt. If this occurs and a device driver for the SMBus controller is not loaded by the OS, system lockup can occur. To set the controller to a proper state, System BIOS should write FFh to the Host Status register, SMBBASE + 00h.

17.7

SMBus 2.0 Support

PCH provides a SMBus 2.0 host controller. The Packet Error Checking (PEC) protocol is supported. PCH can execute transactions with either PEC enabled or disabled using the PEC_EN bit, SMB_BASE + 02h[7].

When this bit is a 1, the host controller will perform transactions with the packet error checking phase appended. This bit must be written prior to the write in which the "START" bit is set.

- For writes, the value of the PEC byte is transferred from the PEC Register.



- For reads, the value of the PEC byte is loaded into PEC Register.

The Packet Error Check (PEC) Register, SMB_BASE + 08h, contains the 8-bit CRC value that is used as the Packet Error check on the SMBus. This capability allows for more reliable transfers on the SMBus. For writes, this register is written by software prior to running the command. For reads, this register is read by software after the read command is completed on SMBus.

There are two mechanisms available in the PCH component for PEC byte handling. Only one mechanism has to be enabled by the software.

- Hardware assist mechanism for PEC Byte transfers – PCH calculates the PEC Byte content for the transfer when the AAC (SMB_BASE + 0Dh[0]) bit is set.
- Software mechanism – Software has the responsibility to calculate the PEC Byte content for the transfer when PEC_EN bit is set.

Steps for a PEC Enabled Transfer (Write):

1. Clear the I2C_EN bit.
2. Clear the PEC_EN bit.
3. Set the AAC bit.
4. Load the SMBus Address, SMBus Transfer Count, SMBus Transfer Data and the SMBus Command.
5. Start the SMBus Host.
6. The steps for checking the Host Busy bit and the successful transfers are verified.
7. Clear the Byte Done Status bit at the end of the transfer.

17.8 SMBus Block Transfers and TCO

The System BIOS should always clear the SECOND_TO_STS bit (TCO2_STS Register, TCOBASE + 6h[1]) before executing any SMBus Block Reads or Writes.

17.9 Support for Handling Incomplete SMBus Transactions

PCH provides support for handling conditions on the SMBus where a transaction was not completed on the SMBus, and simple SMBus devices are waiting for the transaction completion, and the system is unable to release the SMBus data line until the required number of clock cycles have been issued to complete the transaction. An incomplete SMBus transaction can happen for a variety of reasons, including such occasions as a system reset occurrence in the middle of a SMBus master transaction on the bus. It is recommended that the System BIOS supports handling incomplete SMBus transactions, especially when using the SMBus after a system reset for such items as reading a DIMM's SPD.

The incomplete transaction that can be handled by this mechanism consists of the condition where a device is holding the SMBus data pin low while the SMBus clock pin is high. This condition will typically result in the HOST_BUSY bit in the HST_STS register remaining set for a time period greater than the maximum time period that may occur for the SMBus transaction requested. This condition is a result of 2 possible scenarios:



1. The SMBus is truly busy and the PCH has not detected an idle condition. Other devices on the SMBus are continuously using the bus. This is a possible though very unlikely scenario.
2. A SMBus device has experienced an incomplete transaction, and is currently driving the SMBDATA line low.

Software can detect the presence of an incomplete transaction by using the SMBus_PIN_CTL register to observe SMBCLK at a high state for > 50us while SMBDATA is low during that same time period. It is assumed that software attempting to detect or clear this transaction owns the INUSE_STS semaphore.

If software determines an incomplete transaction is present on the SMBus, it can follow the basic algorithm for handling incomplete SMBus transactions described below:

1. Kill the current SMBus host controller transaction by setting the KILL bit, SMBus + 02h[1].
2. Force the SMBus clock low by clearing the SMBCLK_CTL bit, SMBus + 0Fh[2].
3. Delay for 50us.
4. Release the SMBus clock pin, allowing it to return to a high state by setting the SMBCLK_CTL bit.
5. Read the SMBus clock state by reading the SMBCLK_CUR_STS bit, SMBus + 0Fh[0], and if it is not set, then another device is stretching the SMBus clock. Repeat step 5 until the SMBCLK_CUR_STS bit set or tTIMEOUT, MAX (35ms) has passed, at which point the incomplete transaction cannot be cleared and this process must be aborted.
6. Delay for 50us.
7. Repeat steps 2 through 6 630 times to complete the longest possible transaction.
8. If the FAILED bit, SMBus + 00h[4] is set, then the bus was successfully cleared.

17.10 SMBus System BIOS Responsibilities

System BIOS should ensure proper register programming for the S3, S4, S5, and G3 software execution paths. These register settings (shown in the table below) must be set at the time of BIOS handoff to the OS (Int 19h or OS waking vector) in order to ensure proper OS load and driver startup.

Register.Field	Value	Comments
PCI offset 2Ch-2Eh	OEM	These registers must be set to a non-zero value of the OEM's assigned system vendor ID and system device ID.
PCI offset 40h[1]	0	Unless the SMI handler is using the SMBus controller by setting the INUSE_STS semaphore, this bit must be 0.
SMBBASE + 02h[0]	0	Unless the SMI handler is using the SMBus controller by setting the INUSE_STS semaphore, this bit must be 0.
SMBBASE + 11h[2]	X	BIOS must set this bit to the proper value to allow generation of SMI# or interrupts due to SMBALERT#



Register.Field	Value	Comments
SMBBASE + 11h[1]	X	BIOS must set this bit to the proper value according to the platform's support of host notify wake events.
SMBBASE + 11h[0]	X	BIOS must set this bit to the proper value to allow generation of SMI# or interrupts according to Host Notify.

17.11 SMI/OS Driver Resource Arbitration

System designs may request that the PCH supports a usage model that allows both a present OS driver and a runtime SMI handler to make use of the SMBus controller. The system management mode and OS runtime are in completely separate processor execution contexts, and do not share a common memory resource that can be used for communication and synchronization between the OS driver and SMI. Without synchronization and SMI/Driver communication, unpredictable results can occur in the environment when a change is made in one context that affects the operation of the other. An example could be one context accessing a SMBus controller register at the same time as the other context, or more complex in nature in that one context may change the address assignment of a device the other context was using. This section will cover the areas of resource arbitration: I/O space ownership and interrupt ownership.

17.11.1 SMBus Host Controller Ownership

The I/O space of the PCH SMBus controller contains I/O space for three SMBus devices: a SMBus host controller, a SMBus slave device, and a host notify slave. After OS boot, this specification partitions responsibility (and control) of these three SMBus devices between the OS driver and SMI as follows:

Category	Registers	Owner
Standard PCI Configuration Header Space	Vendor ID, Device ID, Command, Device Status, Class Code, Base Address, SVID, SID, Interrupt Line	OS PCI driver
Device Specific PCI Configuration Space	Host Configuration	IN_USE owner (SMI or OS Device Driver) ¹
Host Master	Host Status, Host Control, Host Command, Transmit Slave Address, Host Data 0, Host Data 1, Block Data Byte, Packet Error Check, Auxiliary Control, SMBus Pin Control	IN_USE owner (SMI or OS Device Driver) ²
TCO Slave	Receive Slave Address, Slave Data, SMLink Pin Control	SMI Only
Host Notify Slave	Slave Status, Slave Command, Notify Device Address, Notify Data Low Byte, Notify Data High Byte	OS Driver ³

1. The SMI handler must save the previous value of the Host_Configuration register value prior to modifying this register when it owns the INUSE_STS semaphore. Prior to releasing the INUSE_STS semaphore, SMI must restore the contents of this register.



2. The SMI handler may modify host specific control registers when it owns the INUSE_STS bit.
3. The SMI handler must disable the HOST_NOTIFY_INTREN, SMBus + 11[0], and SMBALERT#, SMBus + 11h[2], interrupts in the Slave Command register prior to using the host controller in SMI. The Slave_Command register must be restored to its original value prior to releasing the INUSE_STS semaphore.

The TCO Slave SMBus device is solely responsible for IA FW SMI functionality; at no time should the present OS driver modify the contents of the registers.

The Host Master and Host Notify Slave registers can be used by either runtime SMI software or OS present driver. Both the SMI handler and OS driver must obey the I/O ownership arbitration mechanism described in this section.

The APL provides a simple mechanism for software in two different execution contexts to arbitrate for ownership of the I/O resources of the AOL SMBus host controller: the INUSE_STS bit (bit 6 of offset 0h of the Host Status register). The INUSE_STS bit is a hardware semaphore bit that can be used to determine that the SMBus host controller I/O space is currently available for use by software reading the bit.

Software must obtain ownership of the host controller via the INUSE_STS semaphore prior to performing reads or writes to the registers listed in the table above. The Host_Status register can be read at any time, as it is part of the INUSE_STS semaphore ownership arbitration mechanism.

The SMI handler or OS driver must obtain ownership of the SMBus host registers prior to modifying their contents. After the transaction is complete, and ownership can be yielded to the other context without side effect, the INUSE_STS bit should be cleared by writing a 1 to that bit without disturbing the value of other bits in the Host Status register. Software can do this by writing a value of 40h to the Host Status I/O port offset. A completed transaction may be the completion (interrupt assertion) of a single instance of a SMBus protocol, or may require completion of many SMBus bus protocol packets in order to complete an autonomous sequence on the SMBus, such as a SMBus ARP sequence. Therefore, a SMBus sequence may take over several seconds to complete. Steps to obtain ownership of the SMBus Host Controller I/O space are as follows:

1. Read the Host Status register.
2. If INUSE_STS is clear, then ownership is obtained. Software should set a flag (DriverOwned or SmiOwned, depending on context) = TRUE to indicate it owns the SMBus host controller.
3. If INUSE_STS is set, then the SMBus host functionality is currently in use by the other software context. Software (SMI or OS driver) should schedule a periodic call back, and again arbitrate for ownership by starting again at step 1, when the call back is serviced.

Software owning the SMBus host controller may now initiate SMBus transaction(s). Software should assume all registers shared between the OS driver and SMI (Host Status, Host Control, Cost Command, Transmit Slave Address, Host Data 0, Host Data 1, Block Data Byte, Packet Error Check, Auxiliary Control, and SMBus Pin Control) are in an unknown state, and initialize all necessary registers prior to performing a transaction. The only exception is the interrupt enables, which must be saved and restored by SMI. After completion of the transaction(s), software should clear the INUSE_STS bit by writing a value of 1 to INUSE_STS.



Note: The owning software entity does not need to release ownership of the SMBus host controller registers after every SMBus transaction; it may hold SMBus host controller ownership through multiple transactions to ensure an autonomous operation to a SMBus device or through a SMBus transaction sequence such as ARP. However, it is recommended that software entities release ownership of the SMBus after every autonomous operation to allow timely opportunity to other system software.

17.11.2 SMBus Host Controller Interrupt Ownership

The OS PCI Driver assumes it has full ownership of the SMBus host controller interrupts during system runtime. Prior to the System BIOS passing control to the OS boot loader, the System BIOS must set the following configuration bits to the following values:

Register Bit	Value	Comment
PCI offset 40h [1]	0	Interrupts are always routed to the OS PCI driver
SMBus + 02h [0]	0	Interrupts on command completion are disabled.
SMBus + 011h [2]	1	Interrupts from SMBAlert# pin are disabled.
SMBus + 011h [0]	0	Interrupts from host notify commands are disabled.

These settings will ensure that no interrupts can occur between System BIOS releasing control of the SMBus host controller to the OS and when the OS SMBus controller driver starts can occur. The consequences of such an interrupt are unpredictable, but will most likely result in a system hang as no software is installed to handle the interrupt.

The System BIOS must follow a very strict model in order to perform a SMBus transaction using SMI during runtime. The System BIOS must only modify the SMBus I/O registers when it has obtained ownership of the SMBus host controller registers. Furthermore, System BIOS must restore previous interrupt configuration after it has used the SMBus in order to maintain software transparency from existing OS present software. The sequence of System BIOS operations would proceed in the following manner:

1. SMI context is entered.

SMI software decides it needs to run a SMBus transaction.

SMI software arbitrates for ownership of the SMBus host controller I/O space via the INUSE_STS bit. Note that this may require exit and re-entry of SMI in order for OS present software to release ownership of the INUSE_STS bit. SMI software can do this using either the periodic SMI timer source or the software SMI timer source, depending on the urgency of the SMBus access.

SMI software saves the state of the INTREN, SMBALERT_DIS, and HOST_NOTIFY_INTREN bits.

SMI software disables all SMBus interrupts by programming INTREN to 0, SMBALERT_DIS to 1, and HOST_NOTIFY_INTREN to 0.

SMI switches the SMBus interrupt to SMI by setting PCI offset 40h[1].

SMI configures the SMBus host controller registers for the SMBus transaction. At this time, the SMI handler may set INTREN to allow SMI interrupt on transaction completion.



SMI may exit back to OS context and be re-entered when the transaction is completed with an SMI. Note that the INUSE_STS bit should still be set as the SMI still owns the SMBus host controller I/O space.

SMI handles the SMBus transaction completion interrupt, and may at this time repeat steps 7-9 until it is finished using the SMBus host controller.

SMI switches the SMBus interrupt back to PCI-based interrupt by clearing PCI offset 40h[1].

SMI software restores the state of the INTREN, SMBALERT_DIS, and HOST_NOTIFY_INTREN bits to the values saved in step 4.

SMI software releases ownership of the SMBus I/O registers by writing a 1 to Host_Status.INUSE_STS bit.

Note: The SMI handler must keep asynchronous interrupts disabled. These are Slave_Command.SMBALERT_DIS = 1, and Slave_Command.HOST_NOTIFY_INTREN = 0. During the SMI's SMBus transaction, these interrupts cannot be serviced as they require use of the SMBus. SMBAlert# requires reading the ARA address, which requires a SMBus transaction which cannot be performed while a transaction is already in progress. A SMBus Host Notify can only occur due to a protocol transaction on the SMBus, so will not occur while another transaction is proceeding on the bus.

Note: The System BIOS must be aware that the I/O base address register and I/O space enable registers may be changed (or disabled) at any time by the PCI bus driver of the operating system. The System BIOS must handle this situation by reserving an alternate I/O space location that the SMI handler can map the SMBus controller to if it desires SMBus access during the time the PCI bus driver may have disabled the I/O space of the SMBus controller.

17.12 SMBus Handling for System Resets

During a SMBus transaction in which an I²C device is sending information to the SoC, the I²C device may not release the SMBus if the SoC receives an asynchronous reset. For example, if a system reset occurs while reading a DIMM's EEPROM, then the PCH SMBus controller will be reset but the DIMM' EEPROM may not be reset. The EEPROM continues with the Byte Read protocol even though the SMBus host has been reset to a default state. When the first SMBus transaction is attempted after the reset, the PCH SMBus controller may hang with the HOST_BUSY bit set.

When the above condition occurs, the SMBus will remain hung until the slave device is reset. Usually this requires a cold system reset, or other means of resetting the slave device.

The system BIOS should incorporate both of the following mechanisms to overcome such a scenario:

For any warm reset, cold reset, or system power down that is detectable by the System BIOS, the System BIOS should ensure that no SMBus transactions are active or pending and take control of the SMBus prior to allowing the reset event to occur. This will reduce exposure to this hang condition.

- a. Read the host status register until HOST_BUSY (bit-0) is deasserted, setting INUSE_STS (bit-6).
- b. For multi-threaded environments, issue a SMBus Kill transaction by setting the Host Control Register (bit-1).



- c. Perform requested CF9h reset or power down to the system.
1. Before the first SMBus attempt in the POST sequence, the System BIOS should detect the failure (hang) condition by attempting a Quick Read protocol to an unused SMBus address. If the HOST_BUSY bit remains set for more than 100ms, then the SMBus is considered hung. When this failure condition is detected, the System BIOS initiates a full system reset by writing port CF9h = 0Ah, followed by writing port CF9h = 0Eh. The following sample code represents one possible implementation of this workaround.

```
; SMBus hang avoidance starts
    mov dx, SMBusBase           ; Host Status Register (00)
    mov al, 05Eh                 ; Clear all status bits
    out dx, al
    add dl, 4                   ; Host Address
Register (04)
    mov al, 0FFh                ; Set Unused SMBus Addr FFh
    out dx, al
    sub dl, 2                   ; Host Control
Register (02)
    mov al, 040h                ; Quick Read Protocol and Start
    out dx, al                  ; command

; wait up to 100 msec for HOST_BUSY to be deasserted
    mov cx, 10000
checkBusy:
    xor eax, eax
    mov al, 10
    JMP_DI DelayUS             ; Delay 10usec
    Mov dx, SMBusBase           ; Host Status Register (00)
    In al, dx                   ; Read host status
register
    Test al, 01h                ; Check BUSY bit
    Jz @f                       ; Response or
host timeout OK
    Loop checkBusy

; do cold reset to overcome a hung SMBus
    mov dx, 0CF9h               ; load dx with reset control reg
    mov al, 0Ah                  ; set bit 3 and bit
1
    out dx, al                  ; write it
    or al, 0Eh                  ; for HRST now set
bit 2
    out dx, al                  ; write it

SMBusAvoidHalt:
    hlt
    jmp SMBusAvoidHalt

@@:
    mov al, 05Eh                ; Clear all status bits
    out dx, al

; SMBus hang avoidance ends
```

```
;-----;  
; DelayUS ;  
;-----;  
; Purpose: Delay input time in microseconds ;  
; Setup: ACPI I/O base address initialized and ACPI I/O space  
enabled.  
; Input: eax # of microseconds to delay ;  
; Output: NONE ;  
; Uses: eax, ebx, esi, dx, di ;  
;-----;
```

Developers should note there are some limitations with using these mechanisms:

1. Use of the alternate SMBus clocking mechanism (SMBus_PIN_CTL, bit-2) on PCH is not recommended because other SMBus slave devices may respond to the stale data as a new transaction.
2. This workaround must occur prior to the first SMBus access in the POST sequence.
3. 100ms timeout specified is valid for a SMBus clock operating at 3.2KHz or higher.
4. This workaround does not take ASF requirements into account.

§



18 High Precision Event Timer (HPET)

APL SoC includes a High Precision Event Timer (HPET) implementation of 1 timer block with 3 comparators. There is only a single possible memory address ranges for HPET registers:

HPET_BASE: FED0_0000h

Refer APL SoC EDS for more details and for a description of the HPET operating modes.

18.1 HPET Enabling

HPET can be disabled or blocked in the following ways:

- HPET_BASE+0x10[0] is the global HPET enable .
- Independent interrupts can be enabled via per-counter enable bit for each timer respectively. HPET_BASE+0x100[0], HPET_BASE+0x120[0], HPET_BASE+0x140[0] are the control registers respectively, and bit 2 is the interrupt enable.
- A PMC bit can be set to ignore HPET before going to S0ix (ETR: ignore_hpet, PMC: 0x48 bit 21).

18.2 High Precision Event Timer Description Table (HPET)

The IA FW has to build the HPET table if the usage of HPET by an OS is desired. HPET is a means to report the Base Addresses of each Event Timer Block early in the OS boot process.

HPET is needed to allow Operating Systems to detect event timers and establish basic timer services for driver load. For a description of the HPET table, refer the IA-PC HPET (High Precision Event Timers) Specification.

18.3 Interrupt Mapping

APL SoC supports only two interrupt mapping modes:

- Legacy option (GC.LRE is set) –
 - Timer 0 is routed to IRQ0 (PIC 8259) / IRQ2 (IOAPIC)
 - Timer 1 is routed to IRQ8
 - Timer 2 is routed according to T2C register
- Standard option (GC.LRE is cleared) – each timer has its own routing control according to TnC registers.

APL SoC does not support “FSB option” (direct MSI)



18.4 HPET ASL Code

The following ASL code is required by the IA FW to report resource consumption for HPET:

```
Device(HPET) // High Precision Event Timer
{
    Name (_HID, EisaId ("PNP0103"))
    Name (_UID, 0x00)
    Method (_STA, 0, NotSerialized)
    {
        Return (0x0F)
    }

    Method (_CRS, 0, Serialized)
    {
        Name (RBUF, ResourceTemplate ())
        {
            Memory32Fixed (ReadWrite,
                            0xFED00000, // Address Base
                            0x00000400, // Address Length
                            )
            Interrupt (ResourceConsumer,
                       Level, ActiveHigh, Exclusive, , , )
            {
                0x00000008, // 0xB HPET-2
            }
        })
        Return (RBUF)
    }
}
```

§



19

GPIO AND CFIO Handling

Most of the GPIOs are multiplexed with other alternate functions and default to their alternate functions on reset. The IA FW must select an alternate function (native) or GPIO after reset. If it is selected as GPIO, IA FW will need to program the polarity for the particular platform application. APL SoC External Design Specification (EDS) contains details of the register information for GPIO programming and native function descriptions.

Some GPIO bits might reside in core power well, while others might be in resume well. See individual bit definition in EDS for more detail.

19.1

GPIO Controller

Apollo Lake GPIO controllers handle all GPIO/CFIO interface to APL SoC. For Apollo Lake, there are four GPIO Communities: North Community, NorthWest Community, West Community and Southwest Community; each community includes “Families” of pads. Each family consists of few “PAD Controllers” and some common blocks. The PAD Controller includes a specific PAD and its attributes, configuration registers, functional capabilities. Each of the family supports PINS and the PINS can be configured as CFIO'S, GPIOs. Once it is configured as GPIO, each family has PINS that can be configured as wake capable, Direct INT capable via IOAPIC or they can be configured as shared INT via 1 INT line for each controller to IOAPIC.

- North Community for DFx GPIO, SATA GPIO, PWM, LPSS/ISH UARTs, IUnit GPIO, JTAG, and SVID.
- Northwest Community for Display GPIO, PMC, Audio, and SPI.
- West Community for LPSS/ISH I2C, ISH GPIO, iCLK, and PMU.
- Southwest Community for EMMC, SDIO, SDCARD, SMBUS and LPC.

19.2

GPIO and IA FW Dependencies

Major role of IA FW are:

- GPIO/PIN, interrupt configuration, during boot up via an IA FW GPIO driver.
- ACPI to enumerate the GPIO Controllers to OS.
- Define GPIO based ACPI event handlers.
- ACPI Opregion support manipulate the GPIO pins from ACPI RTD3 methods.
- Most of the GPIOs are multiplexed with other alternate functions and default to their alternate functions on reset. The IA FW must select an alternate function (native) or GPIO after reset.

IA FW GPIO driver have to perform the following role during Stage-1 phase of the execution.

- Change the Pad Configuration Registers to configure each pad's function, standby state, termination, glitch filter, and interrupt settings.



- For pins used in a functional mode the GPIO Enable should be set to '0' and the Pad Mode should be set to choose the correct function.
- For pins used in GPIO mode, the GPIO Enable should be set to '1', the GPIO Config should be set to configure the pin input and output modes and if output is enabled the TX state should be set using "GPIO TX State".
- For pins that are used as interrupts or wakes the configuration of the glitch filter and interrupt select line must be done to prevent spurious interrupts/wakes.
- Termination is set to either weak pull up or pulls down for any input or unused pin. This is to avoid a floating pin.

19.2.1 Pad Function Changing

All control of pad muxing and GPIO modes is done through the GPIO registers

The BIOS will need to do the following after power up:

- Change the Pad Configuration Registers to configure each pad's function, standby state, termination, glitch filter, and interrupt settings.
- For pins used in a functional mode the GPIO Enable should be set to '0' and the Pad Mode should be set to choose the correct function.
- For pins used in GPIO mode, the GPIO Enable should be set to '1', the GPIO Config should be set to configure the pin input and output modes and if output is enabled the TX state should be set using "GPIO TX State".
- For pins that are used as interrupts or wakes the configuration of the glitch filter and interrupt select line must be done to prevent spurious interrupts/wakes.
- Termination is set to either weak pull up or pull down for any input or unused pin. This is to avoid a floating pin.

19.2.2 GPIO Mode

- All pins controlled by the GPIO SIP are capable of software control through the Pad Configuration Register.
- To enable this mode set GPIO Mode to 1, and then select the desired GPIO Mode.
- For inputs the RX data is available in GPIO RX State portion of Pad Configuration Registers.
- For outputs the TX control is done through the GPIO TX state bits

19.2.3 GPIO Interrupt

The GPIO SIP supports the following types of interrupts:

- Falling Edge
- Rising Edge
- Level Sensitive Interrupt.

Follow the steps below to configure a pin as GPI interrupt



1. First configure the pad to be a GPIO input and set pull up/down as appropriate to prevent spurious interrupts.
2. Next is to set glitch filter. For level sensitive interrupts use a setting which enables filters for RX Data (2 or 3). For edge sensitive use a setting that enables filters on Edges (1 or 3).
3. Next set Interrupt and Wake Configuration for desired interrupt type. For active low level interrupts then invert the RX Data (i.e. InvRXTX).
4. Next choose which interrupt the GPIO is assigned to using Interrupt Sel.
5. Last unmask interrupt in Interrupt Mask Register (0 = masked, 1 = unmasked).

19.2.4 GPIO Wake

For each pad capable of making an SCI/GPE event, there is a field named GPIOIRoutSCI in the per-pad DWO register that is used to enable the SCI event generation. All SCI enabled pad trigger outputs are ORed together and sent out as a single wire to the PMC. Internally the PMC ORs the five community wires together into a single wake event.

When the PMC sees an SCI/GPE wake event, it will wake the system and send an SCI to the host. The host will then send a sideband read to the ACPI register that will be proxied to the appropriate GPIO GPI_SPE_STS based on the GPE0_DW* mapping, which the PMC is also aware of.

19.3 GPIO Registers

Most of the GPIOs are multiplexed with other alternate modes. Each GPIO may have maximum 16 modes. The IA FW must program and select mode according to the platform requirement after reset. If it is selected as GPIO, IA FW will need to program the polarity, interrupt, terminators etc for the particular platform application. APL SoC EDS contains details of the register information for GPIO programming and native function descriptions.

Some GPIO bits might reside in core power well, while others might be in resume well. See individual bit definition in EDS for more detail.

APL SoC GPIOs divide into 4 communities and each community have a few families. GPIO registers in each GPIO community are accessible through IOSF-SB DWORD-aligned byte address .Following are the details.

19.3.1 GPIO Memory Space Register

Community Base address for GPIO community memory space registers is accessible through IOSF-SB with below port ID.

IOSF-SB PORT ID	Description	Registers Length
0xCO	GPIO Southwest Community	0x4F8
0xC4	GPIO Northwest Community	0x640



IOSF-SB PORT ID	Description	Registers Length
0xC5	GPIO North Community	0x698
0xC7	GPIO West Community	0x550

19.3.2 Community Registers

Following table describes registers description and address offset according to pad number and its CFIO controller:

Table 18-1. Per Pad Memory Space Registers Addresses

Register Name	MMIO Address	Description
READ_ACCESS_POLICY	COMMUNITY_BASE + 0x0000	Read Access Control
WRITE_ACCESS_POLICY	COMMUNITY_BASE + 0x0100	Write Access Control
WAKE_STATUS_REG	COMMUNITY_BASE + 0x0200	Wake status register
WAKE_MASK_REG	COMMUNITY_BASE + 0x0280	Wake Mask register
INT_STATUS_REG	COMMUNITY_BASE + 0x0300	GPIO Interrupt Status
INT_MASK_REG	COMMUNITY_BASE + 0x0380	GPIO Interrupt mask
PAD_CFG0	COMMUNITY_BASE + 0x4400 + 0x400 * FAMILY_NUMBER + 8 * PAD_NUMBER	Pad Control Register 0
PAD_CFG1	COMMUNITY_BASE + 0x4404 + 0x400 * FAMILY_NUMBER + 8 * PAD_NUMBER	Pad Control Register 1
FAMILY_RCOMP_CTRL	COMMUNITY_BASE + 0x1080 + 0x80 * family#	RCOMP Control Register
FAMILY_RCOMP_OFFSET	COMMUNITY_BASE + 0x1084 + 0x80 * family#	RCOMP Offset Register
FAMILY_RCOMP_OVERRIDE	COMMUNITY_BASE + 0x1088 + 0x80 * family#	RCOMP Override Register
FAMILY_RCOMP_VALUE	COMMUNITY_BASE + 0x108C + 0x80 * family#	RCOMP Value Register
FAMILY_CONF_RCOMP	COMMUNITY_BASE + 0x1090 + 0x80 * family#	Family config rcomp register
FAMILY_CONF_REG	COMMUNITY_BASE + 0x1094 + 0x80 * family#	GPIO family configuration register

Where:

COMMUNITY_BASE: Get from D13:F0:0x10

19.4 Serial GPIO Feature

APL SoC does not support the serial GPIO feature.

19.5 GPIO Interrupt

There are two approaches to programming a GPIO to cause an interrupt. DirectIRQ and SharedIRQ.

Shared IRQ:

1. Controlled by status and enable registers: GPI_INT_STS, GPI_INT_EN
2. Assert sent on IRQ 14/15

Direct IRQ:

1. No wake support from S0ix on direct IRQ enabled GPIO interrupt
2. Each pad has a different IRQ number, interrupts number between 0x19-0xBA

All GPIO communities are recommended to use common shared IRQ interrupt, GPIO driver should read 'GP_INT_STS' register for each of communities to determine the interrupt source.

19.6 GPIO Interrupt and Wake Capabilities

All GPIOs can be configured as either input or output. All pins that default to GPIO mode input have their output drivers tri-stated including during power-on and reset. If a certain pin state is desired to be maintained on the GPIO mode input pins, external pull up or down is needed. All pins that default as GPIO outputs of 0 level shall not glitch high during power-on and reset.

The following tables show GPIO capabilities, refer Apollo Lake EDS Volume1 for details on default and native functions, directions and all the available functions.

GPIO	GPIO # in Community	Native Function(s)	SMI	Direct IRQ	Direct IRQ Number	Direct IRQ Wake Wire	SCI Tier 1 Group
Southwest Community							
GPIO_205	0	PCIE_WAKE0_B	Y	N	n/a	N	0
GPIO_206	1	PCIE_WAKE1_B	Y	N	n/a	N	0
GPIO_207	2	PCIE_WAKE2_B	Y	N	n/a	N	0
GPIO_208	3	PCIE_WAKE3_B	Y	N	n/a	N	0



GPIO	GPIO # in Community	Native Function(s)	SMI	Direct IRQ	Direct IRQ Number	Direct IRQ Wake Wire	SCI Tier 1 Group
GPIO_156	4	EMMCO_CLK	N	N	n/a	N	0
GPIO_157	5	EMMCO_D0	N	N	n/a	N	0
GPIO_158	6	EMMCO_D1	N	N	n/a	N	0
GPIO_159	7	EMMCO_D2	N	N	n/a	N	0
GPIO_160	8	EMMCO_D3	N	N	n/a	N	0
GPIO_161	9	EMMCO_D4	N	N	n/a	N	0
GPIO_162	10	EMMCO_D5	N	N	n/a	N	0
GPIO_163	11	EMMCO_D6	N	N	n/a	N	0
GPIO_164	12	EMMCO_D7	N	N	n/a	N	0
GPIO_165	13	EMMCO_CMD	N	N	n/a	N	0
GPIO_166	14	SDIO_CLK	N	N	n/a	N	0
GPIO_167	15	SDIO_D0	Y	N	n/a	N	0
GPIO_168	16	SDIO_D1	N	N	n/a	N	0
GPIO_169	17	SDIO_D2	N	N	n/a	N	0
GPIO_170	18	SDIO_D3	N	N	n/a	N	0
GPIO_171	19	SDIO_CMD	N	N	n/a	N	0
GPIO_172	20	SDCARD_CLK	N	N	n/a	N	0
GPIO_179	21	SDCARD_CLK_FB	N	N	n/a	N	0
GPIO_173	22	SDCARD_D0	Y	N	n/a	N	0
GPIO_174	23	SDCARD_D1	N	N	n/a	N	0
GPIO_175	24	SDCARD_D2	N	N	n/a	N	0
GPIO_176	25	SDCARD_D3	N	N	n/a	N	0
GPIO_177	26	SDCARD_CD_B	N	N	n/a	N	0
GPIO_178	27	SDCARD_CMD	N	N	n/a	N	0
GPIO_186	28	SDCARD_LVL_WP	N	N	n/a	N	0
GPIO_182	29	EMMCO_STROBE	N	N	n/a	N	0
GPIO_183	30	SDIO_PWR_DOWN_B	Y	N	n/a	N	0
SMB_ALERTB	31	SMB_ALERTB	N	N	n/a	N	0
SMB_CLK	32	SMB_CLK	N	N	n/a	N	1



GPIO	GPIO # in Community	Native Function(s)	SMI	Direct IRQ	Direct IRQ Number	Direct IRQ Wake Wire	SCI Tier 1 Group
SMB_DATA	33	SMB_DATA	N	N	n/a	N	1
LPC_ILB_SERIRQ	34	LPC_ILB_SERIRQ	N	N	n/a	N	1
LPC_CLKOUT0	35	LPC_CLKOUT0	N	N	n/a	N	1
LPC_CLKOUT1	36	LPC_CLKOUT1	N	N	n/a	N	1
LPC_ADO	37	LPC_ADO	N	N	n/a	N	1
LPC_AD1	38	LPC_AD1	N	N	n/a	N	1
LPC_AD2	39	LPC_AD2	N	N	n/a	N	1
LPC_AD3	40	LPC_AD3	N	N	n/a	N	1
LPC_CLKRUNB	41	LPC_CLKRUNB	N	N	n/a	N	1
LPC_FRAMEB	42	LPC_FRAMEB	N	N	n/a	N	1
West Community							
GPIO_124	0	LPSS_I2C0_SDA	Y	N	n/a	N	2
GPIO_125	1	LPSS_I2C0_SCL	Y	N	n/a	N	2
GPIO_126	2	LPSS_I2C1_SDA	Y	N	n/a	N	2
GPIO_127	3	LPSS_I2C1_SCL	Y	N	n/a	N	2
GPIO_128	4	LPSS_I2C2_SDA	Y	N	n/a	N	2
GPIO_129	5	LPSS_I2C2_SCL	Y	N	n/a	N	2
GPIO_130	6	LPSS_I2C3_SDA	Y	N	n/a	N	2
GPIO_131	7	LPSS_I2C3_SCL	Y	N	n/a	N	2
GPIO_132	8	LPSS_I2C4_SDA	Y	N	n/a	N	2
GPIO_133	9	LPSS_I2C4_SCL	Y	N	n/a	N	2
GPIO_134	10	LPSS_I2C5_SDA	Y	N	n/a	N	2
GPIO_135	11	LPSS_I2C5_SCL	Y	N	n/a	N	2
GPIO_136	12	LPSS_I2C6_SDA	Y	N	n/a	N	2
GPIO_137	13	LPSS_I2C6_SCL	Y	N	n/a	N	2
GPIO_138	14	LPSS_I2C7_SDA	Y	N	n/a	N	2
GPIO_139	15	LPSS_I2C7_SCL	Y	N	n/a	N	2
GPIO_146	16	ISH_GPIO_0	Y	N	n/a	N	2
GPIO_147	17	ISH_GPIO_1	Y	N	n/a	N	2



GPIO	GPIO # in Community	Native Function(s)	SMI	Direct IRQ	Direct IRQ Number	Direct IRQ Wake Wire	SCI Tier 1 Group
GPIO_148	18	ISH_GPIO_2	Y	N	n/a	N	2
GPIO_149	19	ISH_GPIO_3	Y	N	n/a	N	2
GPIO_150	20	ISH_GPIO_4	Y	N	n/a	N	2
GPIO_151	21	ISH_GPIO_5	Y	N	n/a	N	2
GPIO_152	22	ISH_GPIO_6	Y	N	n/a	N	2
GPIO_153	23	ISH_GPIO_7	Y	N	n/a	N	2
GPIO_154	24	ISH_GPIO_8	Y	N	n/a	N	2
GPIO_155	25	ISH_GPIO_9	Y	N	n/a	N	2
GPIO_209	26	PCIE_CLKREQ0_B	N	N	n/a	N	N
GPIO_210	27	PCIE_CLKREQ1_B	N	N	n/a	N	N
GPIO_211	28	PCIE_CLKREQ2_B	N	N	n/a	N	N
GPIO_212	29	PCIE_CLKREQ3_B	N	N	n/a	N	N
OSC_CLK_OUT_0	30	OSC_CLK_OUT_0	N	N	n/a	N	N
OSC_CLK_OUT_1	31	OSC_CLK_OUT_1	N	N	n/a	N	N
OSC_CLK_OUT_2	32	OSC_CLK_OUT_2	N	N	n/a	N	N
OSC_CLK_OUT_3	33	OSC_CLK_OUT_3	N	N	n/a	N	N
OSC_CLK_OUT_4	34	OSC_CLK_OUT_4	N	N	n/a	N	N
PMU_AC_PRESENT	35	PMU_AC_PRESENT	N	N	n/a	N	N
PMU_BATLOW_B	36	PMU_BATLOW_B	N	N	n/a	N	N
PMU_PLTRST_B	37	PMU_PLTRST_B	N	N	n/a	N	N
PMU_PWRBTN_B	38	PMU_PWRBTN_B	N	N	n/a	N	N
PMU_RESETBUTTON_B	39	PMU_RESETBUTTON_B	N	N	n/a	N	N
PMU_SLP_SO_B	40	PMU_SLP_SO_B	N	N	n/a	N	N
PMU_SLP_S3_B	41	PMU_SLP_S3_B	N	N	n/a	N	N
PMU_SLP_S4_B	42	PMU_SLP_S4_B	N	N	n/a	N	N
PMU_SUSCLK	43	PMU_SUSCLK	N	N	n/a	N	N
PMU_WAKE_B	44	PMU_WAKE_B	N	N	n/a	N	N
SUS_STAT_B	45	SUS_STAT_B	N	N	n/a	N	N
SUSPWRDNACK	46	SUSPWRDNACK	N	N	n/a	N	N



GPIO	GPIO # in Community	Native Function(s)	SMI	Direct IRQ	Direct IRQ Number	Direct IRQ Wake Wire	SCI Tier 1 Group
Northwest Community							
GPIO_187	0	HV_DDI0_DDC_SDA	N	N	n/a	Y	4
GPIO_188	1	HV_DDI0_DDC_SCL	N	N	n/a	Y	4
GPIO_189	2	HV_DDI1_DDC_SDA	N	N	n/a	Y	4
GPIO_190	3	HV_DDI1_DDC_SCL	N	N	n/a	Y	4
GPIO_191	4	DBI_SDA	N	N	n/a	Y	4
GPIO_192	5	DBI_SCL	N	N	n/a	Y	4
GPIO_193	6	PANEL0_VDDEN	N	N	n/a	Y	4
GPIO_194	7	PANEL0_BKL滕	N	N	n/a	Y	4
GPIO_195	8	PANEL0_BKLTCTL	N	N	n/a	Y	4
GPIO_196	9	PANEL1_VDDEN	N	N	n/a	Y	4
GPIO_197	10	PANEL1_BKL滕	N	N	n/a	Y	4
GPIO_198	11	PANEL1_BKLTCTL	N	N	n/a	Y	4
GPIO_199	12	DBI_CSX	N	N	n/a	Y	4
GPIO_200	13	DBI_RESX	N	N	n/a	Y	4
GPIO_201	14	GP_INTD_DSI_TE1	Y	N	n/a	Y	4
GPIO_202	15	GP_INTD_DSI_TE2	Y	N	n/a	Y	4
GPIO_203	16	USB_OCO_B	N	N	n/a	Y	4
GPIO_204	17	USB_OC1_B	N	N	n/a	Y	4
PMC_SPI_FSO	18	PMC_SPI_FSO	N	N	n/a	Y	N
PMC_SPI_FS1	19	PMC_SPI_FS1	N	N	n/a	Y	N
PMC_SPI_FS2	20	PMC_SPI_FS2	N	N	n/a	Y	N
PMC_SPI_RXD	21	PMC_SPI_RXD	N	N	n/a	Y	N
PMC_SPI_TXD	22	PMC_SPI_TXD	N	N	n/a	Y	N
PMC_SPI_CLK	23	PMC_SPI_CLK	N	N	n/a	Y	N
PMIC_PWRGOOD	24	PMIC_PWRGOOD	N	N	n/a	Y	N
PMIC_RESET_B	25	PMIC_RESET_B	N	N	n/a	Y	N
GPIO_213	26	PMIC_SDWN_B	N	N	n/a	Y	N
GPIO_214	27	PMIC_BCUDISW2	N	N	n/a	Y	N



GPIO	GPIO # in Community	Native Function(s)	SMI	Direct IRQ	Direct IRQ Number	Direct IRQ Wake Wire	SCI Tier 1 Group
GPIO_215	28	PMIC_BCUDISCRIT	N	N	n/a	Y	N
PMIC_THERMTRIP_B	29	PMIC_THERMTRIP_B	N	N	n/a	Y	N
PMIC_STDBY	30	PMIC_STDBY	N	N	n/a	Y	N
PROCHOT_B	31	PROCHOT_B	N	N	n/a	Y	N
PMIC_I2C_SCL	32	PMIC_I2C_SCL	N	N	n/a	Y	N
PMIC_I2C_SDA	33	PMIC_I2C_SDA	Y	Y	32h	Y	5
GPIO_74	34	AVS_I2S1_MCLK	Y	Y	33h	Y	5
GPIO_75	35	AVS_I2S1_BCLK	Y	Y	34h	Y	5
GPIO_76	36	AVS_I2S1_WS_SYNC	Y	Y	35h	Y	5
GPIO_77	37	AVS_I2S1_SDI	Y	Y	36h	Y	5
GPIO_78	38	AVS_I2S1_SDO	Y	Y	37h	Y	5
GPIO_79	39	AVS_M_CLK_A1	Y	Y	38h	Y	5
GPIO_80	40	AVS_M_CLK_B1	Y	Y	39h	Y	5
GPIO_81	41	AVS_M_DATA_1	Y	Y	3Ah	Y	5
GPIO_82	42	AVS_M_CLK_AB2	Y	Y	3Bh	Y	5
GPIO_83	43	AVS_M_DATA_2	N	Y	3Ch	Y	5
GPIO_84	44	AVS_I2S2_MCLK	N	Y	3Dh	Y	5
GPIO_85	45	AVS_I2S2_BCLK	N	Y	3Eh	Y	5
GPIO_86	46	AVS_I2S2_WS_SYNC	N	Y	3Fh	Y	5
GPIO_87	47	AVS_I2S2_SDI	N	Y	40h	Y	5
GPIO_88	48	AVS_I2S2_SDO	N	Y	41h	Y	5
GPIO_89	49	AVS_I2S3_BCLK	N	Y	42h	Y	5
GPIO_90	50	AVS_I2S3_WS_SYNC	N	Y	43h	Y	5
GPIO_91	51	AVS_I2S3_SDI	N	Y	44h	Y	5
GPIO_92	52	AVS_I2S3_SDO	N	N	n/a	Y	N
GPIO_97	53	FST_SPI_CS0_B	N	Y	49h	Y	5
GPIO_98	54	FST_SPI_CS1_B	N	Y	4Ah	Y	5
GPIO_99	55	FST_SPI_MOSI_IO0	N	Y	4Bh	Y	5
GPIO_100	56	FST_SPI_MISO_IO1	N	Y	4Ch	Y	5



GPIO	GPIO # in Community	Native Function(s)	SMI	Direct IRQ	Direct IRQ Number	Direct IRQ Wake Wire	SCI Tier 1 Group
GPIO_101	57	FST_SPI_IO2	N	Y	4Dh	Y	5
GPIO_102	58	FST_SPI_IO3	N	Y	4Eh	Y	5
GPIO_103	59	FST_SPI_CLK	N	Y	4Fh	Y	5
FST_SPI_CLK_FB	60	FST_SPI_CLK_FB	N	N	n/a	Y	N
GPIO_104	61	GP_SSP_0_CLK	N	Y	50h	Y	5
GPIO_105	62	GP_SSP_0_FS0	N	Y	51h	Y	5
GPIO_106	63	GP_SSP_0_FS1	N	Y	52h	Y	5
GPIO_109	64	GP_SSP_0_RXD	N	Y	54h	Y	6
GPIO_110	65	GP_SSP_0_TXD	N	Y	55h	Y	6
GPIO_111	66	GP_SSP_1_CLK	N	Y	56h	Y	6
GPIO_112	67	GP_SSP_1_FS0	N	Y	57h	Y	6
GPIO_113	68	GP_SSP_1_FS1	N	Y	58h	Y	6
GPIO_116	69	GP_SSP_1_RXD	N	Y	5Bh	Y	6
GPIO_117	70	GP_SSP_1_TXD	N	Y	5Ch	Y	6
GPIO_118	71	GP_SSP_2_CLK	Y	Y	5Dh	Y	6
GPIO_119	72	GP_SSP_2_FS0	Y	Y	5Eh	Y	6
GPIO_120	73	GP_SSP_2_FS1	Y	Y	5Fh	Y	6
GPIO_121	74	GP_SSP_2_FS2	Y	Y	60h	Y	6
GPIO_122	75	GP_SSP_2_RXD	Y	Y	61h	Y	6
GPIO_123	76	GP_SSP_2_TXD	Y	Y	62h	Y	6
North Community							
GPIO_0	0	TRACE_O_CLK_VNN	Y	Y	63h	Y	7
GPIO_1	1	TRACE_O_DATA0_VNN	Y	Y	64h	Y	7
GPIO_2	2	TRACE_O_DATA1_VNN	Y	Y	65h	Y	7
GPIO_3	3	TRACE_O_DATA2_VNN	Y	Y	66h	Y	7
GPIO_4	4	TRACE_O_DATA3_VNN	Y	Y	67h	Y	7
GPIO_5	5	TRACE_O_DATA4_VNN	Y	Y	68h	Y	7
GPIO_6	6	TRACE_O_DATA5_VNN	Y	Y	69h	Y	7
GPIO_7	7	TRACE_O_DATA6_VNN	Y	Y	6Ah	Y	7



GPIO	GPIO # in Community	Native Function(s)	SMI	Direct IRQ	Direct IRQ Number	Direct IRQ Wake Wire	SCI Tier 1 Group
GPIO_8	8	TRACE_0_DATA7_VNN	Y	Y	6Bh	Y	7
GPIO_9	9	TRACE_1_CLK_VNN	Y	Y	6Ch	Y	7
GPIO_10	10	TRACE_1_DATA0_VNN	Y	Y	6Dh	Y	7
GPIO_11	11	TRACE_1_DATA1_VNN	Y	Y	6Eh	Y	7
GPIO_12	12	TRACE_1_DATA2_VNN	Y	Y	6Fh	Y	7
GPIO_13	13	TRACE_1_DATA3_VNN	Y	Y	6Fh	Y	7
GPIO_14	14	TRACE_1_DATA4_VNN	Y	Y	71h	Y	7
GPIO_15	15	TRACE_1_DATA5_VNN	Y	Y	72h	Y	7
GPIO_16	16	TRACE_1_DATA6_VNN	Y	Y	73h	Y	7
GPIO_17	17	TRACE_1_DATA7_VNN	Y	Y	74h	Y	7
GPIO_18	18	TRACE_2_CLK_VNN	Y	Y	75h	Y	7
GPIO_19	19	TRACE_2_DATA0_VNN	Y	Y	76h	Y	7
GPIO_20	20	TRACE_2_DATA1_VNN	Y	Y	77h	Y	7
GPIO_21	21	TRACE_2_DATA2_VNN	Y	Y	32h	Y	7
GPIO_22	22	TRACE_2_DATA3_VNN	Y	Y	33h	Y	7
GPIO_23	23	TRACE_2_DATA4_VNN	Y	Y	34h	Y	7
GPIO_24	24	TRACE_2_DATA5_VNN	Y	Y	35h	Y	7
GPIO_25	25	TRACE_2_DATA6_VNN	Y	Y	36h	Y	7
GPIO_26	26	TRACE_2_DATA7_VNN	Y	Y	37h	Y	7
GPIO_27	27	TRIGOUT_0	Y	Y	38h	Y	7
GPIO_28	28	TRIGOUT_1	Y	Y	39h	Y	7
GPIO_29	29	TRIGIN_0	Y	Y	3Ah	Y	7
GPIO_30	30	ISH_GPIO_12	Y	Y	3Bh	Y	7
GPIO_31	31	ISH_GPIO_13	Y	Y	3Ch	Y	7
GPIO_32	32	ISH_GPIO_14	Y	Y	3Dh	Y	8
GPIO_33	33	ISH_GPIO_15	Y	Y	3Eh	Y	8
GPIO_34	34	PWM0	N	Y	3Fh	Y	8
GPIO_35	35	PWM1	N	Y	40h	Y	8
GPIO_36	36	PWM2	N	Y	41h	Y	8



GPIO	GPIO # in Community	Native Function(s)	SMI	Direct IRQ	Direct IRQ Number	Direct IRQ Wake Wire	SCI Tier 1 Group
GPIO_37	37	PWM3	N	Y	42h	Y	8
GPIO_38	38	LPSS_UART0_RXD	Y	Y	43h	Y	8
GPIO_39	39	LPSS_UART0_TXD	Y	Y	44h	Y	8
GPIO_40	40	LPSS_UART0_RTS_B	Y	Y	45h	Y	8
GPIO_41	41	LPSS_UART0_CTS_B	Y	Y	46h	Y	8
GPIO_42	42	LPSS_UART1_RXD	Y	Y	47h	Y	8
GPIO_43	43	LPSS_UART1_TXD	Y	Y	48h	Y	8
GPIO_44	44	LPSS_UART1_RTS_B	Y	Y	49h	Y	8
GPIO_45	45	LPSS_UART1_CTS_B	Y	Y	4Ah	Y	8
GPIO_46	46	LPSS_UART2_RXD	Y	Y	4Bh	Y	8
GPIO_47	47	LPSS_UART2_TXD	Y	Y	4Ch	Y	8
GPIO_48	48	LPSS_UART2_RTS_B	Y	Y	4Dh	Y	8
GPIO_49	49	LPSS_UART2_CTS_B	Y	Y	4Eh	Y	8
GPIO_62	50	GP_CAMERASB00	Y	Y	5Bh	Y	8
GPIO_63	51	GP_CAMERASB01	Y	Y	5Ch	Y	8
GPIO_64	52	GP_CAMERASB02	Y	Y	5Dh	Y	8
GPIO_65	53	GP_CAMERASB03	Y	Y	5Eh	Y	8
GPIO_66	54	GP_CAMERASB04	Y	Y	5Fh	Y	8
GPIO_67	55	GP_CAMERASB05	Y	Y	60h	Y	8
GPIO_68	56	GP_CAMERASB06	Y	Y	61h	Y	8
GPIO_69	57	GP_CAMERASB07	Y	Y	62h	Y	8
GPIO_70	58	GP_CAMERASB08	Y	Y	63h	Y	8
GPIO_71	59	GP_CAMERASB09	Y	Y	64h	Y	8
GPIO_72	60	GP_CAMERASB10	Y	Y	65h	Y	8
GPIO_73	61	GP_CAMERASB11	Y	Y	66h	Y	8
TCK	62	TCK	N	N	n/a	Y	N
TRST_B	63	TRST_B	N	N	n/a	Y	N
TMS	64	TMS	N	N	n/a	Y	N
TDI	65	TDI	N	N	n/a	Y	N

GPIO	GPIO # in Community	Native Function(s)	SMI	Direct IRQ	Direct IRQ Number	Direct IRQ Wake Wire	SCI Tier 1 Group
CX_PMODE	66	CX_PMODE	N	N	n/a	Y	N
CX_PREQ_B	67	CX_PREQ_B	N	N	n/a	Y	N
JTAGX	68	JTAGX	N	N	n/a	Y	N
CX_PRDY_B	69	CX_PRDY_B	N	N	n/a	Y	N
TDO	70	TDO	N	N	n/a	Y	N
CNV_BRI_DT	71	CNV_BRI_DT	N	N	n/a	Y	N
CNV_BRI_RSP	72	CNV_BRI_RSP	N	N	n/a	Y	N
CNV_RGI_DT	73	CNV_RGI_DT	N	N	n/a	Y	N
CNV_RGI_RSP	74	CNV_RGI_RSP	N	N	n/a	Y	N
SVIDEO_ALERT_B	75	SVIDEO_ALERT_B	N	N	n/a	Y	N
SVIDEO_DATA	76	SVIDEO_DATA	N	N	n/a	Y	N
SVIDEO_CLK	77	SVIDEO_CLK	N	N	n/a	Y	N

§



20 LPSS (Low Power Sub System) System

The Intel® LPSS block consists of 16 controllers as listed in [Table 20-1](#).

Table 20-1. LPSS Block Controllers

Device:Function	Description
D22:F0	LPSS I2C0 Controller
D22:F1	LPSS I2C1 Controller
D22:F2	LPSS I2C2 Controller
D22:F3	LPSS I2C3 Controller
D23:F0	LPSS I2C4 Controller
D23:F1	LPSS I2C5 Controller
D23:F2	LPSS I2C6 Controller
D23:F3	LPSS I2C7 Controller
D24:F0	LPSS UART0 Controller
D24:F1	LPSS UART1 Controller
D24:F2	LPSS UART2 Controller
D24:F3	LPSS UART3 Controller
D25:F0	LPSS SPI0 Controller
D25:F1	LPSS SPI1 Controller
D25:F2	LPSS SPI2 Controller
D26:F0	LPSS PWM Controller

20.1 Intel® LPSS Initialization Flow

The IA FW initialization flow for each of the Intel® LPSS controller is as follow:

- IA FW may disable the controller prior to PCI enumeration. If disabled then below steps should be skipped.
- IA FW must perform clock programming (not all controllers).
- IA FW must perform controller reset release (not all controllers).
- IA FW must perform the Power Management programming for LPSS devices.

The IA FW must perform the Low Power Feature programming.

The IA FW must enumerate the controller in PCI Mode.



If the device at function 0 is to be disabled, the rest of the functions must be disabled as per the PCI specification on multi-function devices.

Detailed programming procedures are in ConfigureLpssIoDevices function in ScLpss.c of SIC reference package.

20.2 Disabling Intel LPSS Controllers

If Intel® LPSS controllers are enabled by fuse and soft-strap but still need to be function disabled, the IA FW can disable each of the controllers in Intel® LPSS by following the steps below.

1. The IA FW must set BAR0 offset 0x204[3] = 1, where BAR0 is from Dx:Fy:Reg 0x10
2. The IA FW must set BAR0 offset 0x24c[2] = 1, where BAR0 is from Dx:Fy:Reg 0x10
3. The IA FW must set the Dx:Fy:Reg 0x84 [1:0] = 11b, to put the controller in D3 HOT State.

The IA FW must set the Function disable bit of the Dx:Fy as indicated in the table below.

Table 20-2. Function Disable Programming for Each of the LPSS Controller

Note: PBASE is the base address of PMC. Where is located at (Bus 0, dev 0x0d, fun 1, offset 0x10) + 0x1000.

Ex: the value in Bus 0, dev 0x0d, fun 1, offset 0x10 = 0xFE042000

The PBASE will be 0xFE042000 + 0x1000 = 0xFE043000

PBASE + 0x34 is the Function Disable 0 (FUNC_DIS_0) register in PMC

Device:Function	Description	Function Disable Location
D22:F0	LPSS I2C0 Controller	PBASE + 0x34[21]
D22:F1	LPSS I2C1 Controller	PBASE + 0x34[20]
D22:F2	LPSS I2C2 Controller	PBASE + 0x34[19]
D22:F3	LPSS I2C3 Controller	PBASE + 0x34[18]
D23:F0	LPSS I2C4 Controller	PBASE + 0x34[17]
D23:F1	LPSS I2C5 Controller	PBASE + 0x34[16]
D23:F2	LPSS I2C6 Controller	PBASE + 0x34[15]
D23:F3	LPSS I2C7 Controller	PBASE + 0x34[14]
D24:F0	LPSS UART0 Controller	PBASE + 0x34[13]
D24:F1	LPSS UART1 Controller	PBASE + 0x34[12]
D24:F2	LPSS UART2 Controller	PBASE + 0x34[11]
D24:F3	LPSS UART3 Controller	PBASE + 0x34[10]
D25:F0	LPSS SPI0 Controller	PBASE + 0x34[9]



Device:Function	Description	Function Disable Location
D25:F1	LPSS SPI1 Controller	PBASE + 0x34[8]
D25:F2	LPSS SPI2 Controller	PBASE + 0x34[7]
D26:F0	LPSS PWM Controller	PBASE + 0x34[6]

20.3 Intel® LPSS Controller Reset Release

The IA FW must release reset on the following Intel® LPSS controllers for the controller to function properly.

Table 19-3. Reset Release Programming for Each Intel® LPSS Controller

Device:Function	Description
D22:F0	BAR0 + 0x204 [1:0] = 0x3
D22:F1	BAR0 + 0x204 [1:0] = 0x3
D22:F2	BAR0 + 0x204 [1:0] = 0x3
D22:F3	BAR0 + 0x204 [1:0] = 0x3
D23:F0	BAR0 + 0x204 [1:0] = 0x3
D23:F1	BAR0 + 0x204 [1:0] = 0x3
D23:F2	BAR0 + 0x204 [1:0] = 0x3
D23:F3	BAR0 + 0x204 [1:0] = 0x3
D24:F0	BAR0 + 0x204 [1:0] = 0x3
D24:F1	BAR0 + 0x204 [1:0] = 0x3
D24:F2	BAR0 + 0x204 [1:0] = 0x3
D24:F3	BAR0 + 0x204 [1:0] = 0x3
D25:F0	BAR0 + 0x204 [1:0] = 0x3
D25:F1	BAR0 + 0x204 [1:0] = 0x3
D25:F2	BAR0 + 0x204 [1:0] = 0x3
D26:F0	BAR0 + 0x204 [1:0] = 0x3

20.4 Intel® LPSS Power Management Programming

The IA FW requirement for Intel® LPSS power management programming is as follow:

Program PM Control Register portid:0x90, offset:0x1d0 [5:0] = 111111'b

For detailed programming, refer ConfigureLpss, ConfigureLpssIoDevices function in ScLpss.c of SIC reference package.

Table 19-4. Clock Gating Programming for Each Intel® LPSS Controller

Device:Function	Description
D22:F0	Program portid:0x90, offset:0x60c [0] = 0
D22:F1	Program sideband portid:0x90, offset:0x60c[1] = 0
D22:F2	Program sideband portid:0x90, offset:0x60c[2] = 0
D22:F3	Program sideband portid:0x90, offset:0x60c[3] = 0
D23:F0	Program portid:0x90, offset:0x60c[4] = 0
D23:F1	Program sideband portid:0x90, offset:0x60c[5] = 0
D23:F2	Program sideband portid:0x90, offset:0x60c[6] = 0
D23:F3	Program sideband portid:0x90, offset:0x60c[7] = 0
D24:F0	Program sideband portid:0x90, offset:0x610[0] = 0
D24:F1	Program sideband portid:0x90, offset:0x610[1] = 0
D24:F2	Program sideband portid:0x90, offset:0x610[2] = 0
D24:F3	Program sideband portid:0x90, offset:0x610[3] = 0
D25:F0	Program sideband portid:0x90, offset:0x614[0] = 0
D25:F1	Program sideband portid:0x90, offset:0x614[1] = 0
D25:F2	Program sideband portid:0x90, offset:0x614[2] = 0
D25:F3	Program sideband portid:0x90, offset:0x614[3] = 0

20.5 Intel® LPSS ACPI Requirements

The IA FW must provide ACPI tables for each of the Intel® LPSS controllers as per ACPI 5.0 specification.

The IA FW must provide ACPI table(s) for the connected sensor(s) that are connected to the Intel® LPSS controllers.

For ACPI ASL definition Refer the south cluster reference code.

Table describes the example of ACPI ASL code for an I2C1 Controller.



Table 19-5. Sample Code

```
Device(I2C1) {
    Name (_ADR, 0x00160001)
    Name (_DDN, "Intel(R) I2C Controller #1")
    Name (_UID, 2)

    Name (RBUF, ResourceTemplate ()
    {
        //FixedDMA(0x12, 0x2, Width32Bit, )
        //FixedDMA(0x13, 0x3, Width32Bit, )
    })
    Method (_CRS, 0x0, NotSerialized)
    {
        Return (RBUF)
    }
} // Device (I2C1)
```

§



21 Storage Control Cluster

The Intel® SCC (Storage Control Cluster) block consist of 3 controllers as listed below.

Table 20-1. SCC Block Controllers

Device:Function	Description
D27:F0	SCC SD Card Controller
D28:F0	SCC eMMC Controller
D30:F0	SCC SDIO Controller

21.1 Intel® SCC Initialization Flow

The IA FW initialization flow for each of the Intel® SCC controller is as follow:

1. The IA FW may disable the controller prior to PCI enumeration. If disable then below steps should be skipped. For SCC disable flow, refer [chapter 21.2](#) for details.
2. The IA FW must perform the additional programming.
3. The IA FW must perform the Low Power Feature programming for SCC devices.
4. The IA FW must enumerate the controller in PCI Mode.

21.2 Disabling Intel® SCC Devices

If Intel® SCC devices are enabled by fuse and soft-strap but still need to be function disabled, the IA FW can disable Intel® SCC device individually by following the steps below:

- a. Enable Power Gating bits for each SCC PCI device, by setting PCI configuration register OxA0 bit [18 : 17]
- b. Put device in D3Hot state before disabling it, as below sequence:
 1. The Restore required must be cleared by IA FW by writing a '1' to it. Clear SCC BAR0.reg_D0i3 0x81C[3] by writing '1'
 2. SW sets this bit to 1 to move the IP into the D0i3 state. Set SCC BAR0.reg_D0i3 0x81C [2]
 3. Place Device into the D3HOT State by setting 0x084 PMECTRLSTATUS, bits 1:0 (POWERSTATE) to 2'b11.
- c. Set the function disable bit for the SCC device.

21.3 Intel® SCC Additional Programming Steps

- Disable BAR1 for SCC PCI device mode.
- Configure default DLL register Settings for Ax Silicon.
 $SCC_MEM_TX_CMD_DLL_CNTL = 0x505$
 $SCC_MEM_TX_DATA_DLL_CNTL1 = 0xC11$



SCC_MEM_TX_DATA_DLL_CNTL2 = 0x1C2A2927
SCC_MEM_RX_CMD_DATA_DLL_CNTL1 = 0x000D162F
SCC_MEM_RX_STROBE_DLL_CNTL = 0x0a0a
SCC_MEM_RX_CMD_DATA_DLL_CNTL2 = 0x1003b
SCC_MEM_MASTER_DLL_SW_CNTL = 0x001

- Configure default DLL register Settings for APL Bx Silicon.

SCC_MEM_TX_CMD_DLL_CNTL = 0x505
SCC_MEM_TX_DATA_DLL_CNTL1 = 0xA13
SCC_MEM_TX_DATA_DLL_CNTL2 = 0x1C2A2927
SCC_MEM_RX_CMD_DATA_DLL_CNTL1 = 0x000D162F
SCC_MEM_RX_STROBE_DLL_CNTL = 0x0a0a
SCC_MEM_RX_CMD_DATA_DLL_CNTL2 = 0x1003b
SCC_MEM_MASTER_DLL_SW_CNTL = 0x800001

21.4 Intel® SCC ACPI Requirements

The IA FW must provide ACPI table for each of the Intel® SCC as per ACPI 5.0 specification when SCC is working as an ACPI device. For ACPI ASL definition Refer APL reference code. [Table 20-2](#) describes the ACPI ASL code for Intel® SCC.

Table 20-2. Intel® SCC eMMC ACPI ASL Sample Code

```

// SCC eMMC
//
Device (SDHA)
{
    Name (_ADR, 0x001C0000) // _ADR: Address
    Name (_DDN, "Intel(R) eMMC Controller - 80860ACC") // _DDN: DOS
Device Name
    Name (RBUF, ResourceTemplate ())
    {
        }

    OperationRegion(PCCS, PCI_Config, 0x84,0x04)
    Field(PCCS,WordAcc,NoLock,Preserve) {
        PMSR,      32      // 84h: Power Gating Control
    }

    OperationRegion (SCPC, PCI_Config, 0xA0, 4)
    Field (SCPC, WordAcc, NoLock, Preserve)
    {
        Offset (0x00),           // 0xA0 D0i3 Max Power Latency
Powergating Config
        ,          17,
        I3EN, 1,
        DPGE, 1
    }

    Method (_CRS, 0, NotSerialized) // _CRS: Current Resource
Settings
    {
        Return (RBUF)
    }

    Method (_PS0, 0, NotSerialized) // _PS0: Power State 0
    {
        //
        // Disable power gate
        //
        Store (0, \_SB.PCI0.SDHA.DPGE)
        Store (0, \_SB.PCI0.SDHA.I3EN)

        //
        // Clear clock gate
        //
        \_SB.PCI0.SCPG(0,0xFFFFFFFBE) // Clear bit 6 and 0
        Sleep (2)                  // Sleep 2 ms

        //
        // Restore clock gate
        //
        \_SB.PCI0.SCPG(1,0x00000041) // restore bit 6 and 0
    }

    Method (_PS3, 0, NotSerialized) // _PS3: Power State 3
}

```



```
{  
    Store (PMSR, Local0)  
    //  
    // Enable power gate  
    //  
    Store (1, \_SB.PCI0.SDHA.DPGE)  
    Store (1, \_SB.PCI0.SDHA.I3EN)  
}  
Device (EMMD)  
{  
    Name (_ADR, 0x00000008)           // Slot 0, Function 8  
    Method (_RMV, 0, NotSerialized)   // _RMV: Removal Status  
    {  
        Return (0x0)  
    }  
}  
} //Device(SDHA)
```

§



22 ISH (Integrated Sensor Hub)

22.1 Intel® ISH Initialization Flow

The IA FW initialization flow for the Intel® ISH (Integrated Sensor Hub) is as follow:

1. ISH memory allocation and IMR configuration. For this step, refer [chapter 21.3](#)
2. The IA FW may disable the controller prior to PCI enumeration. If disable then below steps should be skipped. For ISH disabling, refer [chapter 21.3](#)
3. The IA FW must perform the Low Power Feature programming for ISH.

The IA FW must enumerate the controller in PCI mode.

22.2 Intel® ISH IMR (Isolated Memory Region) Setup

The Intel® ISH IMR size is stored in TXE, IA FW has to get this size from TXE and then configure the memory allocation by following the steps:

1. Read the value from register R_SEC_MEM_REQ (0x44) in HECI PCI device. The IMR size will be (value and 0xF0000) >> 16.
2. Calculate the total UMA size for TXE usage including Intel® ISH IMR size.
3. Report the total UMA size usage to system by invoking BuildResourceDescriptorHob().

The process mentioned above is one part of memory initialization.

22.3 Disabling Intel® ISH

If Intel® ISH is enabled by fuse and soft-strap but still need to be function disabled, the IA FW can disable the Intel® ISH by following the steps below:

1. Move Intel® ISH to RTD3Hot by programming PMECTRLSTATUS.POWERSTATE, D10:F0:Reg 0x84 [1:0] to 11b.
2. Set the function disable bit for ISH at PBASE + 0x34 [24].

Requirement is that if ISH is function disabled, then it must not be brought out of reset.

1. ISH has corner cases where it may not respond to reset requests if no FW loaded.
2. Concerns around sensors leaking information in security enabled platforms.

The flow is as follows:

1. IA FW will write the appropriate FUNC_DISABLE0 register (Refer below) in the GCR space.
2. IA FW will record information in NVM that it just disabled a function and is requesting reset.
3. IA FW will trigger a COLD RESET.
4. On reset exit if BIOS finds it has already requested reset (from NVM portion) it will clear that information and continue to perform all the appropriate IA FW function disables as above.



5. IA FW will not see ISH on this new reset since PMC kept it in reset.

Note: Once disabled, the only way to re-enable is global reset or cold off followed by cold boot.

22.3.1 PCI Mode of Operation

IA FW is recommended to leave Intel® ISH to operate in PCI mode at D17:F0 so that operating system can discover it when doing PCI scan. It is recommended to disable BAR1 by setting in ISH private configuration space before PCI enumeration. For disable BAR1, it goes through set side band port 0x94, offset 0x200, bit7.

§



23 *SSC (Spread Spectrum Clocking)*

This chapter is intended to serve as a reference for software and firmware development of EMI and RFI clocking features. The method of reducing EMI transmitted at a center frequency is called Spread Spectrum Clocking or SSC.

Spread spectrum clocking for EMI mitigation is supported in APL. The components affected by SSC are primarily the following:

- DDR Memory
- HSIO (USB3, PCIe, eDP, DP, eMMC, SD and SDIO)

Because the interfaces USB3, PCIe, SATA, eDP, DP, eMMC, SD and SDIO share the same LCPLL SSC clock source, there will be single HighSpeed Serial IO SSC enable/disable and % selection BIOS option for these interfaces in BIOS setup menu. Also, there will be another BIOS option for DDR SSC enable. And these SSC parameters will be programmed via either sideband interface or IPC command.

23.1 *SSC Enable*

For Apollo Lake, the default SSC configuration is set to down-spread with amplitude of 0.5% and a modulation period of 32 kHz. Assuming the default SSC profile is desired for a given platform, the configuration required by software is to enable spread spectrum clocking for the subsystem. Enabling SSC is accomplished by writing the "ssc_en" field of the PLL_CR_RW_CONTROL_x register. The pseudo-code below shows the register values that need to be written to enable or disable spread spectrum clocking.

```
# Enable SSC for High Speed IO domains
LCPLL_CR_RW_CONTROL_1.ssc_en = 0x1

# Enable SSC for DDR
LJ1PLL_CR_RW_CONTROL_1.ssc_en = 0x1
```

Corresponding with the SSC settings are 0~-0.5%, 0.1% stepping. IA FW could set DDR SSC related registers via IPC1 command 0xE8, and set HSIO SSC related registers via IOSF-SB port ID 0x99.

Please refer to Apollo Lake SoC SPI and Signed Master Image Profile (SMIP) Programming Guide Chapter 12.1 for relevant details.

23.2 *DDR SSC Registers*

Default values are applied during cold boot and come from the SMIP. If the values need to be updated then the host must initiate an IPC1 command to the PMC with



command encoding of 0xE8 (EMI/RFI support). Using this command the LJ1PLL registers will be updated after a reset sequence.

Register Name: LJ1PLL_CR_RW_CONTROL_1			
Name	LJ1PLL_CR_RW_CONTROL_1		
Long Name	LJ1PLL_RW_CONTROL_1		
Domain	iclk_vnn		
CR-SB Port ID	0x99		
CR-SB Address	0x9924		
Scope	PACKAGE		
SOix Save			
SAI	ICLK_VNN_POLICY_GRP3		
Description	LJPLL CR		
31:2	RW	0x0	reserved
1:1	RW	0x0	ssc_en Spread Spectrum Clocking: spread enable; 0x0=no frequency spreading; 0x1=enable frequency spreading on PLL output clock
0:0	RW	0x0	ssc_en_ovr SSC enable override

Register Name: LJ1PLL_CR_RW_CONTROL_2			
Name LJ1PLL_CR_RW_CONTROL_2 Long Name LJ1PLL_RW_CONTROL_2 Domain iclk_vnn CR-SB Port ID 0x99 CR-SB Address 0x9928 Scope PACKAGE S0ix Save SAI ICLK_VNN_POLICY_GRP3 Description LJPLL CR			
Range	Access Type	Default (reset)	Description
31:12	RW	0x8888	ssc_frac_step Spread Spectrum Clocking: fractional step configuration; fraction of PLL ratio at which to take frequency modulation steps. eg 0x200000 = (2097152/2^20) * refclk freq = 0.125*19.2 = 2.4MHz steps. Spread magnitude is determined by the step size multiplied by the number of steps in the modulation period (see ssc_cyc_to_peak_m1 for steps per modulation period).
11:11	RW	0x0	reserved
10:9	RW	0x0	ssc_mode Spread Spectrum Clocking: spread direction select; 0x0 = down-spread only; 0x1 = up-spread only; 0x2 = center spread, start with down-spread; 0x3 = center spread, start with up-spread
8:0	RW	0x12B	ssc_cyc_to_peak_m1 Spread Spectrum Clocking: spread period configuration; half the number of steps in the modulation period minus 1. Period of modulation is 2*(value+1) multiplied by the step duration (PLL refclk period). eg 0x12B = 2*(299+1) * (1/19.2MHz) = 600 * 52.083ns = 31.25us. Spread magnitude is determined by the step size (integer + fractional) multiplied by the number of steps in the modulation period (see ssc_frac_step and ssc_ratio_step for step size).



Register Name: LJ1PLL_CR_RW_CONTROL_5			
Name	LJ1PLL_CR_RW_CONTROL_5		
Long Name	LJ1PLL_RW_CONTROL_5		
Domain	iclk_vnn		
CR-SB Port ID	0x99		
CR-SB Address	0x9934		
Scope	PACKAGE		
SOix Save			
SAI	ICLK_VNN_POLICY_GRP3		
Description	LJPLL CR		
31:24	RW	0x7D	pll_ratio_int Clock Bending: integer frequency multiplier; refclk frequency * value = PLL output clock frequency; eg 19.2MHz * 125 = 2400MHz
23:0	RW	0x0	pll_ratio_frac Clock Bending: fractional frequency multiplier; shift PLL clock frequency by (value/2^24)*refclk frequency. eg 0x200000 = (2097152/2^24) * refclk freq = 0.125*19.2 = 2.4MHz

23.3 HSIO SSC Registers

Register Name: LCPLL_CR_RW_CONTROL_1			
Name	LCPLL_CR_RW_CONTROL_1		
Long Name	LCPLL_RW_CONTROL_1		
Domain	iclk_vnn		
CR-SB Port ID	0x99		
CR-SB Address	0x9910		
Scope	PACKAGE		
SOix Save			
SAI	ICLK_VNN_POLICY_GRP3		
Description	LCPLL CR		
31:2	RW	0x0	reserved
1:1	RW	0x0	ssc_en Spread Spectrum Clocking: spread enable; 0x0=no frequency spreading; 0x1=enable frequency spreading on PLL output clock
0:0	RW	0x0	ssc_en_ovr SSC enable override

Register Name: LCPLL_CR_RW_CONTROL_2			
Range	Access Type	Default (reset)	Description
31:12	RW	0x8888	ssc_frac_step Spread Spectrum Clocking: fractional step configuration; fraction of PLL ratio at which to take frequency modulation steps. eg 0x200000 = (2097152/2^20) * refclk freq = 0.125*19.2 = 2.4MHz steps. Spread magnitude is determined by the step size multiplied by the number of steps in the modulation period (see ssc_cyc_to_peak_m1 for steps per modulation period).
11:11	RW	0x0	spare Spare
10:9	RW	0x0	ssc_mode Spread Spectrum Clocking: spread direction select; 0x0 = down-spread only; 0x1 = up-spread only; 0x2 = center spread, start with down-spread; 0x3 = center spread, start with up-spread
8:0	RW	0x12B	ssc_cyc_to_peak_m1 Spread Spectrum Clocking: spread period configuration; half the number of steps in the modulation period minus 1. Period of modulation is 2*(value+1) multiplied by the step duration (PLL refclk period). eg 0x12B = 2*(299+1) * (1/19.2MHz) = 600 * 52.083ns = 31.25us. Spread magnitude is determined by the step size (integer + fractional) multiplied by the number of steps in the modulation period (see ssc_frac_step and ssc_ratio_step for step size).



Register Name: LCPLL_CR_RW_CONTROL_5			
Range	Access Type	Default (reset)	Description
31:24	RW	0x7D	pll_ratio_int Clock Bending: integer frequency multiplier; refclk frequency * value = PLL output clock frequency; eg 19.2MHz * 125 = 2400MHz
23:0	RW	0x0	pll_ratio_frac Clock Bending: fractional frequency multiplier; shift PLL clock frequency by (value/2^24)*refclk frequency. eg 0x200000 = (2097152/2^24) * refclk freq = 0.125*19.2 = 2.4MHz

S



24 Firmware Capsule Update

24.1 Firmware Capsule Update (CU)

A modern computing device contains a number of hardware components that have firmware specific to the component. In the life of a system, it is often desirable to update the firmware to a later version. Firmware update is the process of installing the new firmware on the target system. On Apollo Lake, firmware is classified into two categories: system firmware and device firmware. System firmware is the firmware that is in the critical boot path of the platform and provides runtime services to the system as a whole such as IAFW, Intel ME/TXE FW, Intel® ISS FW, EC FW, and so on. Device Firmware Update refers to the update of non-system critical firmware such as Touch Controller FW, Proximity Sensor FW, and so on.

Capsule Update (CU) is delivered as a Capsule. The capsule typically contains all of the executable code and data necessary to update the system firmware image, such as IA FW, TXE, and MCU.

Note: Intel recommends including Intel ME/TXE firmware as part of system firmware for OEM implementation to avoid validation compatibility issues between BIOS and Intel ME/TXE FW.

24.1.1 Overall Capsule Update (CU) Requirements

- Platform must support 'over the air firmware update' and 'Local update' mechanism.
- IA FW shall provide capsule API (using UpdateCapsule() UEFI API) support.
- Platform must support a secure firmware update, must ensure firmware components are authenticated before update
 - Platform FW must ensure IFWI update is possible from within UEFI aka IAFW context only.
 - IA FW must enable WP (Write Protect) on Flash boot partition before running any unauthorized code to prevent non-firmware agent updating the firmware.
 - Capsule = IFWI + EFI driver code needed for FW update. IFWI as a whole must be authenticated before proceeding for the update
 - Platform firmware update scheme must enable firmware Rollback prevention
 - On every new FW update, FW shall ensure boot is healthy to stage where OEM BIOS can enable manual recovery before enforcing SVN update.
- New FW update boot failure – must auto rollback to the previously existing version.
- Shall have auto-recover ability to ensure recovering from an interrupted firmware update process and continue the FW update without any end user intervention.
- Update shall have generic UX interface to provide the status to user.
- Firmware update must ensure some pre-Update check on system battery/storage availability before the update.
- Shall only support the update entire firmware – IAFW assisted device firmware Update is out of scope for Android*.



IA FW Requirements

1. IA FW should support OS image programming to eMMC through DnX.
2. To allow DnX on SPI for internal use, need to have capability to build internal only SPI images with read/write access to CSE for all SPI regions.

BIOS/IAFW implementation note: This process could be a long flow that can take as long as 1 min under worse case scenarios (examples: on 1st boot where file system is being rebuilt, error cases). If BIOS needs to proceed without CSE properly shutting down NVM access, it is strongly recommended to try at least one global reset and restart the IFWI prepare for update process again). It is recommend to have a bread crumb to track to ensure that there is no reboot loop. BIOS/IAFW/Host can choose to blindly update, but there is a risk that CSE is still attempting to write to data (this risk is higher on SPI platforms where CSE can access its data without host proxy).

24.1.2 IA FW Requirements for Capsule Update via UEFI

For system firmware update via UEFI, IA FW must meet following implementation requirements:

1. UpdateCapsule service implementation: UEFI IA FW must be able to handle Capsules with Capsule flags either with combination of "INITIATE_RESET|PERSIST_ACROSS_RESET" or "INITIATE_RESET|PERSIST_ACROSS_RESET|POPULATE_SYSTEM_TABLE"
2. ESRT system table: IA FW must publish a system table to Windows* to identify firmware resources available for update. Refer "Windows* UEFI firmware update platform" Specification from Microsoft* for more details about ESRT table.
3. Necessary checks for hardening firmware update: this includes battery health check, and firmware storage check if required.

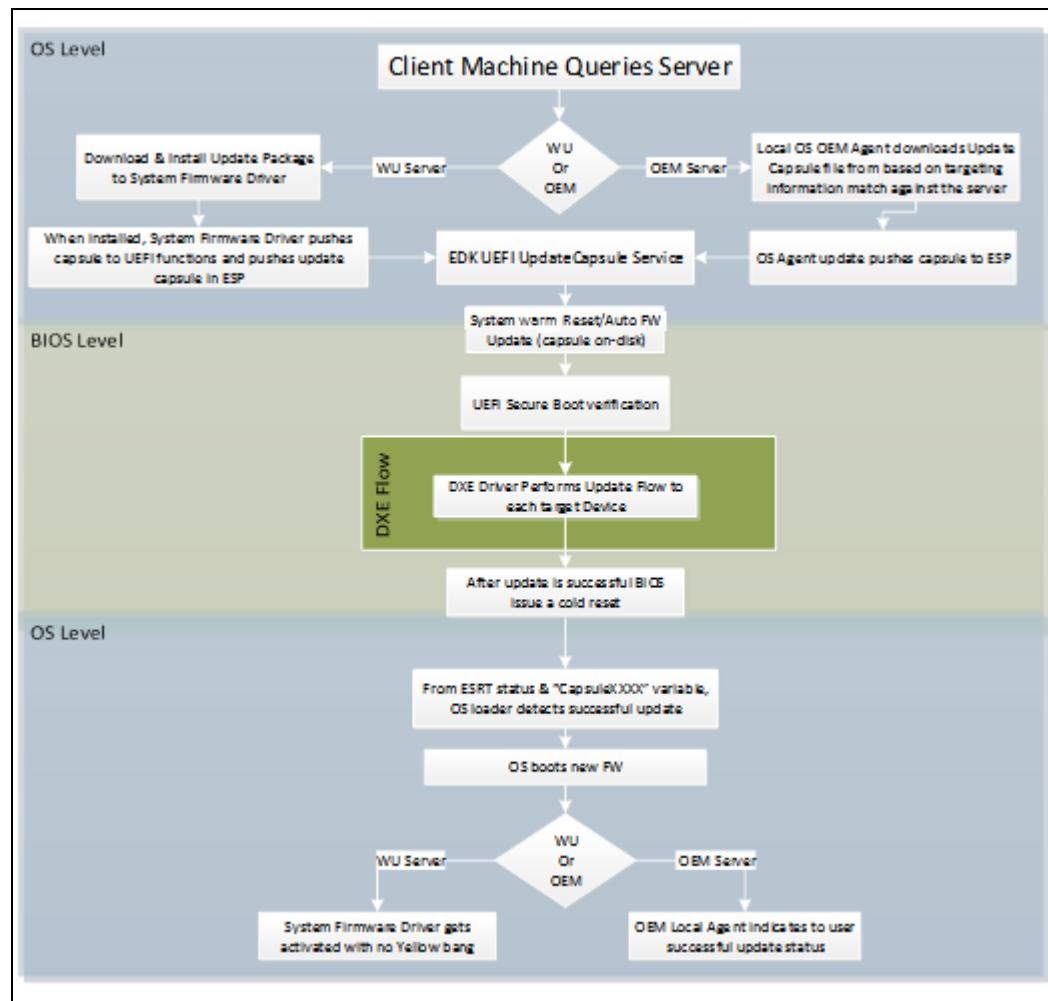
24.1.3 Pushing the Capsule Update Solution Overview

There are two methods to push capsule update:

- 1) Could be achieved using **Windows* Update OR**
- 2) Using **dedicated OEM server** based solution responsible for servicing specific client systems.

Below [Figure 24-1](#) shows an overview of the end-to-end solution flow.

Figure 24-1. OS and BIOS Level Flow of UEFI Capsule Update



24.1.4 Microsoft Windows* Update Mechanism (for Windows* 8.x and above)

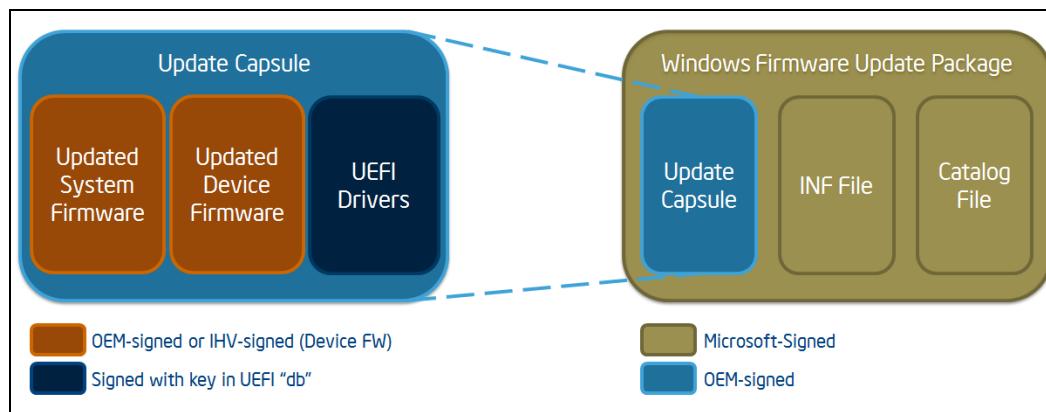
The firmware update process through Windows* update Mechanism is similar to driver update process where in OEM would pack the firmware.bin inside a signed INF file and push it to the end user system.

MSFT classified the firmware into 2 groups; 1- system firmware which is needed for Windows* 8 boot and Windows* 8 basic runtime functions, 2: Device specific firmware for 3rd party devices on Platform.

Each device will have a hardware ID in ESRT table (and there might be multiple IDs for different firmware components for targeted update). The driver package will contain an INF file and an opaque blob. The driver package is signed either by MSFT for Windows* update, or by OEM using authenticode. It packages on IA platforms may Windows* update (WU) mechanism uses an OS UEFI mapped driver. This driver is surfaced up to the OS through an ESRT table entry within the BIOS code. Having this driver allows WU to query version information along with SMBIOS information used to target a specific system make and model against WU server for capsule update mechanism.

Figure below shows the layout of the update capsule and integration within the update package.

Figure 24-2. Windows* Update Firmware Update Package Layout



Summary of each component, in Update Capsule (left):

- Updated System Firmware:** Holds firmware critical to system boot path. This could be BIOS, BIOS + Intel ME/TXE FW, BIOS + Intel ME/TXE FW + EC FW (BIOS vendor implementation dependent) or ISS FW.
- Updated Device Firmware:** Holds firmware for any device. This could be Intel ME/TXE FW, Camera FW, Touch FW, and so on.
- UEFI Drivers:** This is the DXE driver implementing the actual update flow for each of the encapsulated firmware payloads within the capsule. This driver starts execution once the BIOS code finds the capsule and verifies its signature.

Summary of each component, in Windows* Firmware Update Package (right):

- Update Capsule:** This is the update capsule described above.

- **INF File:** This driver provides version check capability for WU to query. Once installed by PnP, this is a driver responsible for pushing the update capsule to ESP. This driver will stay un-activated until OS loader boots new updated FW with ESRT entry indicating successful update.
- **Catalog File:** This signature file is to be signed by Microsoft* (and by OEM for testing).

[Figure 24-3](#) and [Figure 24-4](#) below show an overview of the flow an update package goes through before arriving to the end user system.

Figure 24-3. Windows* Update Mechanism: Update Package High-Level Flow

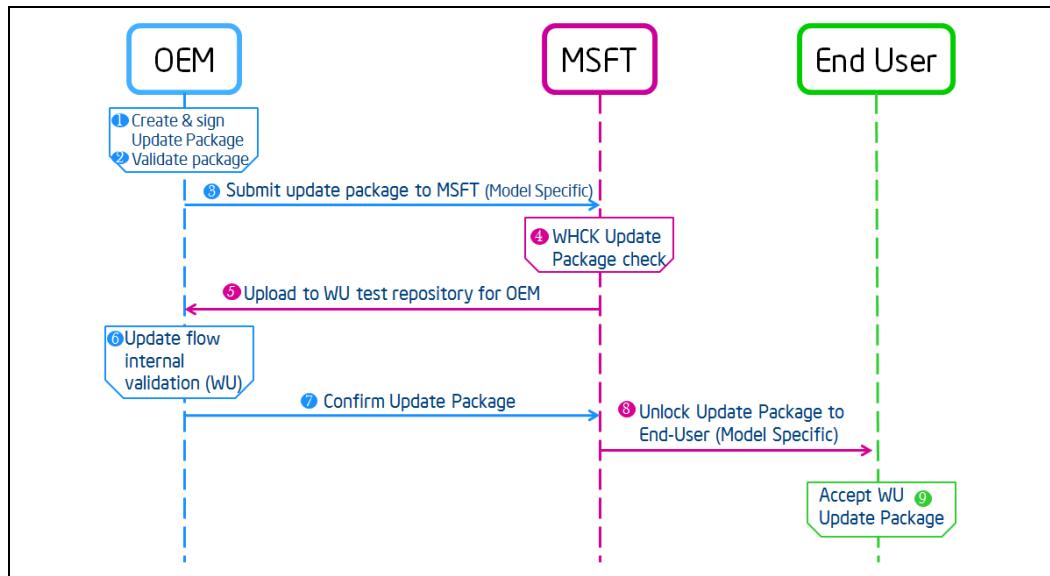
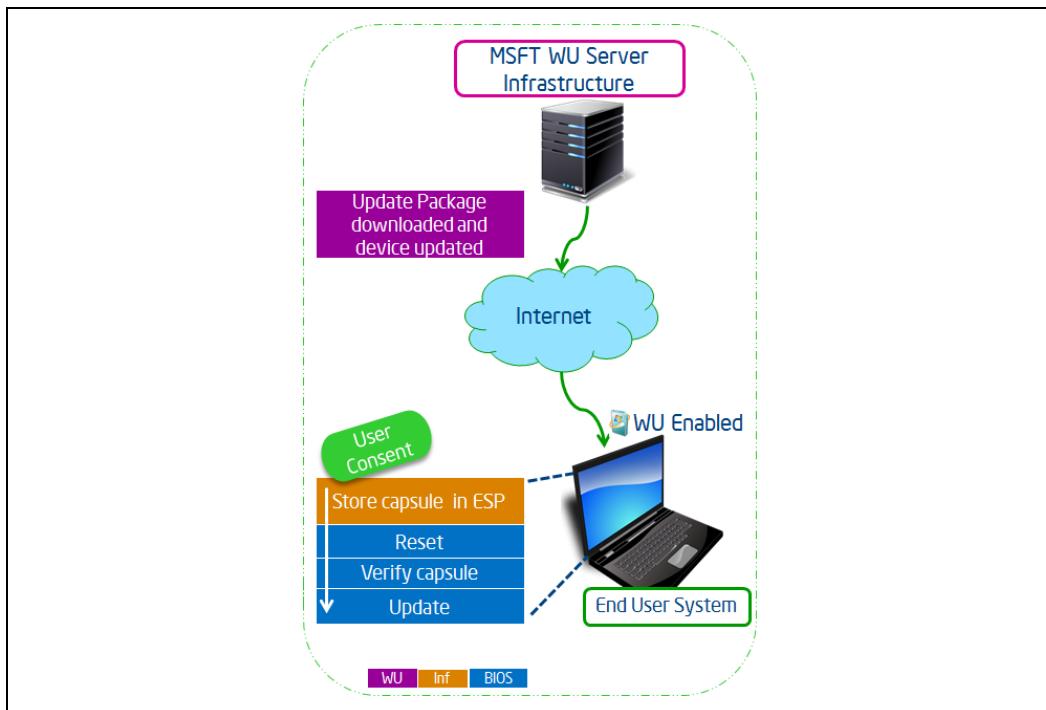


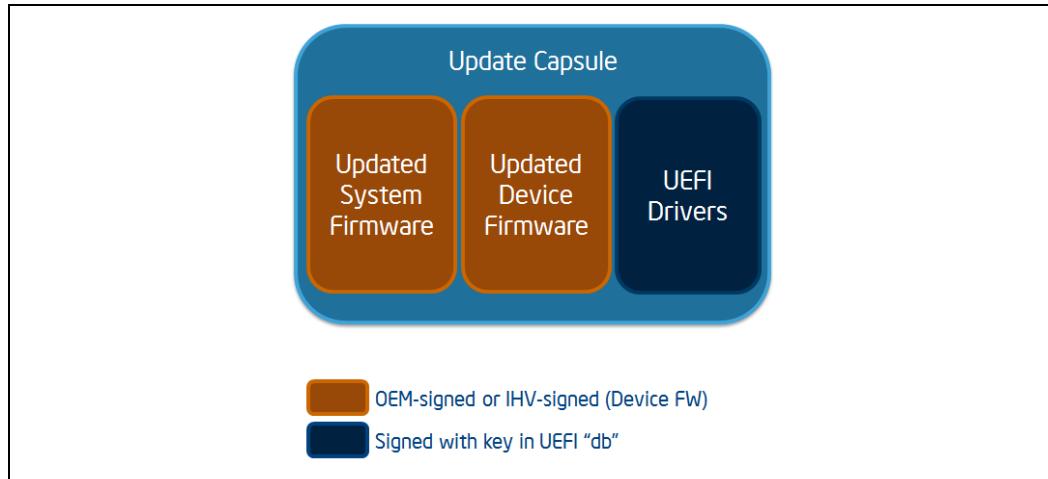
Figure 24-4. Windows* Update Mechanism: End-User View of Update Package Visibility



24.1.5 OEM Update Mechanism

Unlike Windows* update mechanism, OEM Update based solution does not use an OS driver to interface to BIOS. OEM has control to facilitate the means (OEM OS Agent and Server) to directly push the update capsule to ESP. Once capsule exists in ESP, upon reset, BIOS will find it and act upon the encapsulated update flow within the DXE driver.

Figure 24-5. OEM Firmware Update Capsule Layout



Summary of each component, in Update Capsule:

- **Updated System Firmware:** Holds firmware critical to system boot path. This could be BIOS, BIOS + Intel ME/TXE FW, BIOS + Intel ME/TXE FW + EC FW, or ISS FW.
- **Updated Device Firmware:** Holds firmware for any device. This could be Intel ME/TXE FW, ISS FW, Camera FW, Touch FW, and so on.
- **UEFI Drivers:** This is the DXE driver implementing the actual update flow for each of the encapsulated firmware payloads within the capsule. This driver starts execution once the BIOS code finds the capsule and verifies its signature.

[Figure 24-6](#) and [Figure 24-7](#) below show an overview of the flow an update package goes through before arriving to the end user system.

Figure 24-6. OEM Update Mechanism: Update Package High-Level Flow

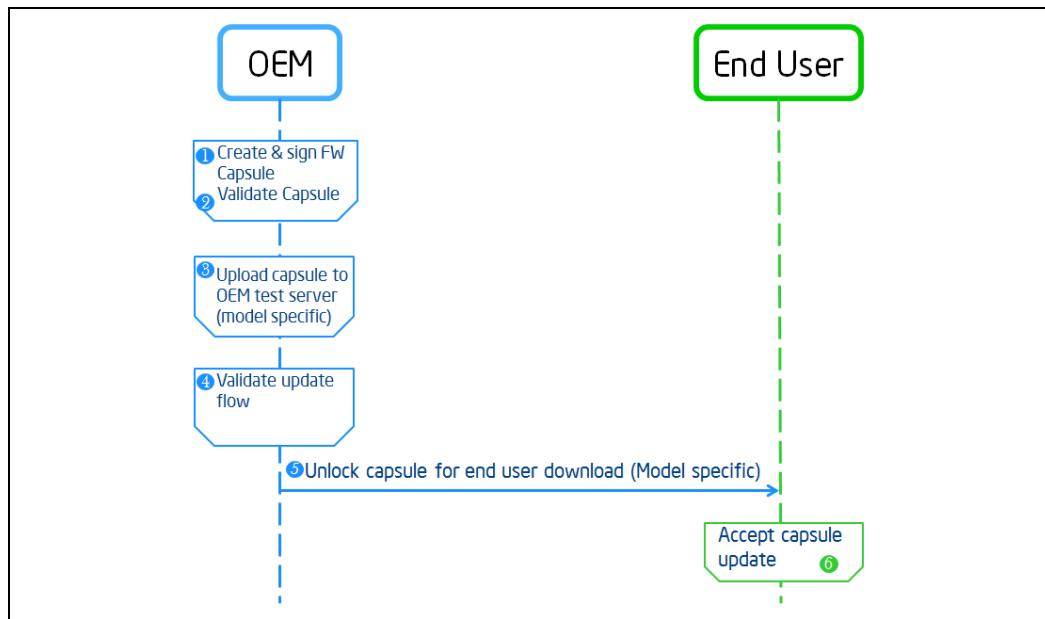
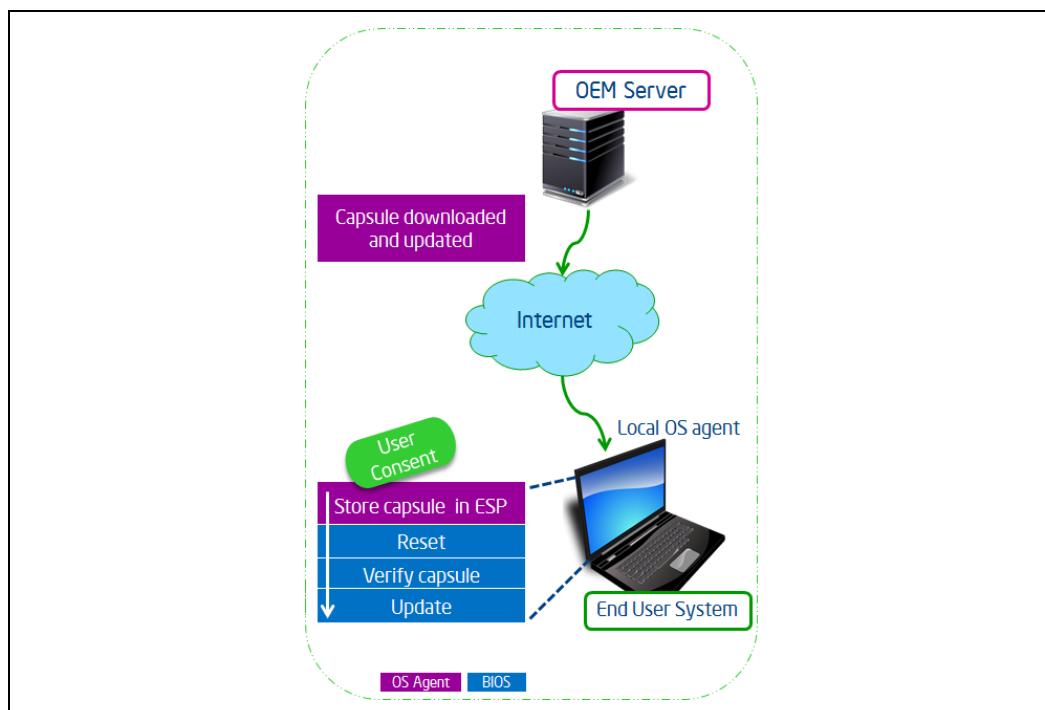


Figure 24-7. OEM Server Update Mechanism: End-User View of Update Package Visibility





24.1.5.1 Local OEM OS Agent

A local OEM agent must be present on the client machine managing key recommendations to facilitate the update capsule to the BIOS ESP. This OEM agent is responsible for pulling the capsule from the OEM managed server; providing the capsule payload to the BIOS as shown above (Refer Chapter 3 OEM Update Mechanism).

24.2 OS Support

The solution in this document supports Windows* 7, Windows* 8.x and Windows* 10. Windows* Update mechanism supports Windows* 8.X and above, while OEM Update mechanism supports Windows* 7, Windows* 8.x and above. Windows* 7 BIOS implementation must be based on UEFI BIOS booting into UEFI mode only.

24.3 OEM Push Server Recommendations

OEM Push based solution requires an OEM server to service each of its models for capsule update. Each model shipped is required to have a local OS agent responsible for querying for available updates. This OS agent is specific for the OEM platform and is owned and implemented by the OEM if they're not depending on Windows* Update for firmware update.

24.3.1 Recommendation of Server-Client Relationship

OEM server should target specific models to allow updates via a client OEM agent existing on target platform. OEM agent is responsible for making the pull of the capsule payload while the server is responsible for:

1. OEM defined method of locking/unlocking updates to specific models.
2. OEM defined method for tracking update adoption data and version information based on acknowledgements from OEM OS agent post updates.
3. OEM defined security and privacy method of OS agent access to server.
4. OEM defined security mechanism and environment for server maintaining the updates.

24.4 OEM OS Agent Recommendation

The local OEM agent has some key requirements to facilitate the update capsule to the target system. In facilitating the update capsule, this agent is responsible for the following:

1. **Privilege:** Require least privilege until an update is necessary (when user approves update).
2. **User Interactions:** OS agent should prompt user when there is critical update necessary for system requiring reset.
3. **Query:** Using OEM defined security and privacy methods; OS agent should securely query the end-user machine against OEM server. Query operation should send SMBIOS information + version information in order to decide if there is an applicable update available. Collection of version information is



OEM defined. Per current implementation of sample agent, agent looks for information describing the capsule for target (Refer note below on security):

- i. Make and Model: SMBIOS information for target.
 - ii. Version information: New FW/BIOS/Capsule version.
 - For each update, Refer VCN for the FW which will not allow roll-back once updating a later VCN number.
 - iii. Hash of capsule payload: SHA256 (or strong) hash used by OEM agent post download to verify download was successful
 - iv. Update criticality: Add capability for OEM to mandate distribution of critical updates while keep high/medium updates optional to end user via interaction at OEM agent service.
4. **Polling:** Polling time should be defined by OEM to periodically check for updates. Agent should also have the capability to allow end user to manually check for updates.
 5. **Targeting:** Utilize SMBIOS to target specific make and model against OEM server.
 6. **Version check:** Based on version checking (OEM method), if newer update capsule exists for a specific model, OS agent should pull the capsule to the end-user machine.
 7. **Push:** OS agent should push the capsule directly to ESP and issue a reset (upon user approval). BIOS would then search and find the capsule upon boot and issue the update flow.
 8. **Pre-Update:** OS agent should perform pre-update preparations (i.e. Suspend BitLocker if enabled immediately before issuing a reset
 9. **Status Check:** OS agent should indicate pass/fail status of capsule update to the user utilizing ESRT table results or "CapsuleXXXX" UEFI spec variable.
 - a. If ESRT table is implemented, after the capsule is successfully installed on user's system, the OS agent can query the capsule update status by looking at the registry key:
Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FirmwareResources\{System Firmware Entry GUID}.
 - i. The LastAttemptStatus field shall be 0, indicating successful firmware capsule update.
 - b. If ESRT table is not implemented, CapsuleXXXX variable can be checked at OS level by OEM agent.
 - i. For UEFI spec 2.4 and onwards.
10. **OEM Stat (optional):** Using OEM defined method, OEM Agent reports back success state post updates.

24.4.1 Agent Communication with OEM Server

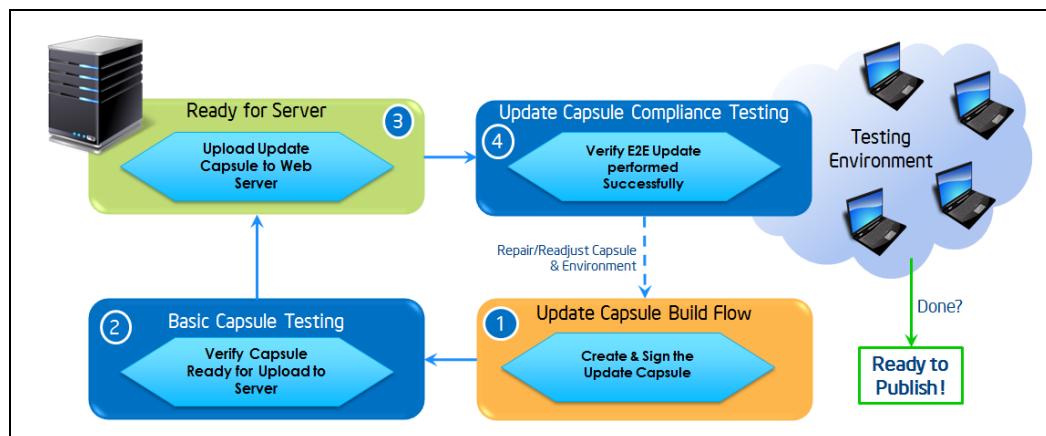
Agent is required to have OEM specific settings and implementation in order to:

1. Query appropriate OEM server for target model.
2. Keep-Alive: Query operation of the OS agent can serve as indication that agent is alive and ready to receive updates.
3. Once update is performed, send update status to server.
4. Polling time should be defined by OEM.

24.5 OEM Push Validation Environment

Before publishing a specific capsule update to a specific model, OEM is responsible for validating the capsule update in a test environment before deploying to all end users as shown below.

Figure 24-8. OEM Validation Environment Reference Flow

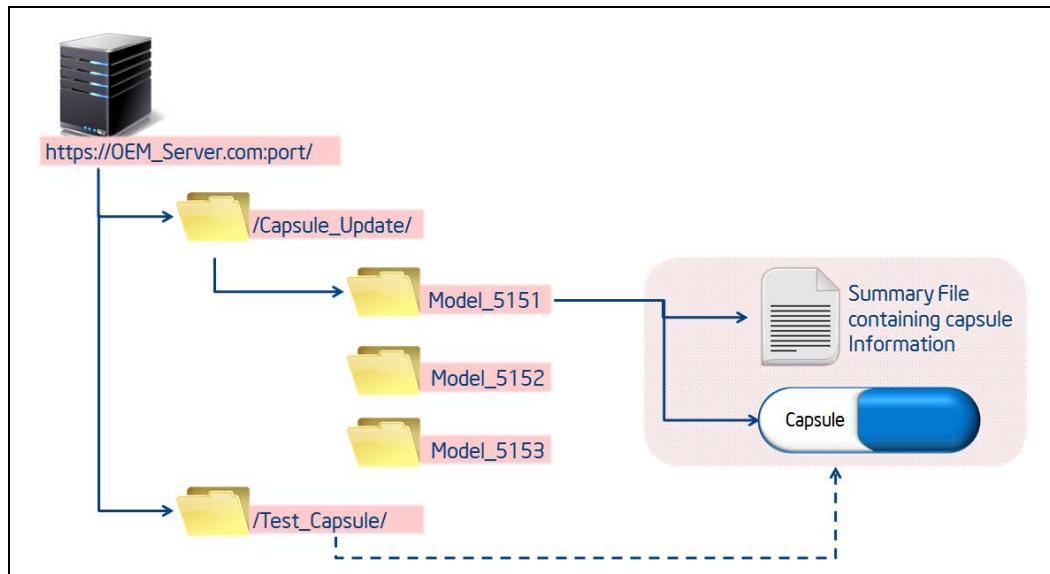


24.5.1 OEM Validation Flow Overview

According to the reference diagram above, OEM should perform the following before deploying a capsule for distribution:

1. **Update Capsule – Build Flow:** Create and Sign the update capsule targeting a specific make and model (Refer FW Update Package Creation chapter for more details).
2. **Basic Capsule Testing:** Manually test the capsule by verifying it is built correctly with all required components (i.e. BIOS and Intel ME/TXE FW, ISS FW where applicable). This can be done by manually placing the capsule in ESP then issuing a reset.
3. **Ready for Server:** Once the capsule is confirmed to be built for a specific target, submit to OEM server for testing before unlocking to all end users. As seen in Figure above, OEM may have multiple models while each managed separately servicing various target platforms. At the server, each model targeted with capsule update would contain:
 - a. Actual capsule file: Capsule containing updated FW/BIOS for target.
 - b. Information file xml describing the capsule.

Figure 24-8. OEM Capsule Update Server High-Level View (Example of what sample is expecting)



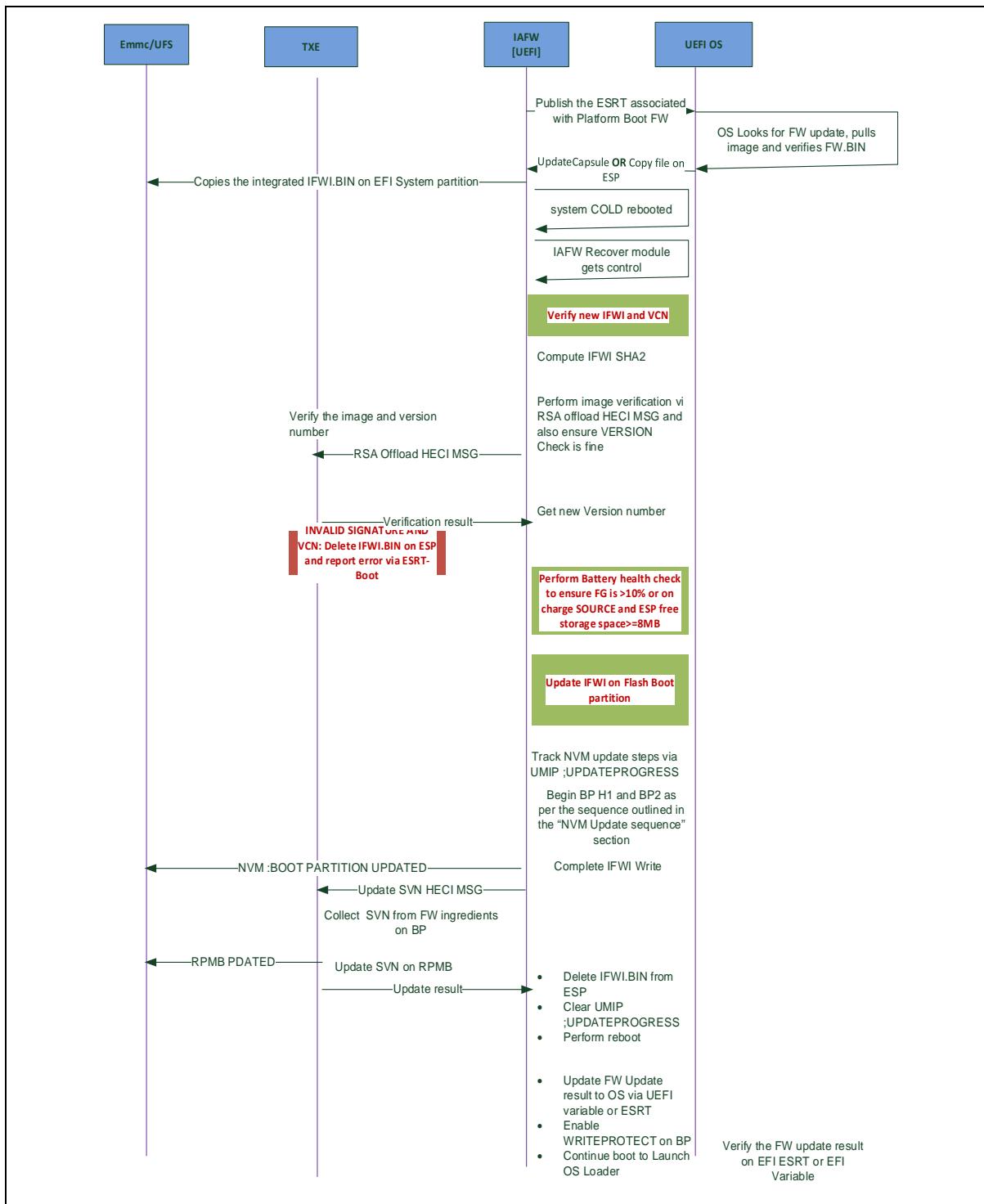
4. **Update Capsule Compliance Testing:** Perform validation to verify server is able to deliver a healthy capsule and client systems are able to receive capsule via tested means.
5. **Ready to Publish:** After validation is complete, mark tested capsule ready to publish for end clients to receive (i.e., Unlocked).



24.6 IAFW Capsule Update Flow Including Interaction with OS and TXE

Overall FW Update Sequence for system firmware update contains these steps:

- OS Capsule update pulling the FW image from the remote server. Refer [Figure 24-6](#) for complete high level flow.
- OS Loader hands over the capsule to IAFW by UpdateCapsule UEFI runtime service.
- System reboots
- Before dispatching into DXE in capsule, IA FW authenticates the capsule.
- IA FW performs pre-checks, including battery life remaining and other protections for the update process.
- In case the capsule is of reset type, IA FW must support persisting the capsule in absence of S3 support.
- DXE driver is dispatched and locates system FW image in the capsule after reset (if required); performs the update and indicate user about the progress if necessary.
- DXE driver installs the protocol to report update status to UpdateCapsule service.
- UpdateCapsule service records the update status in ESRT and commit the update if succeed.
- IAFW tracks and completes the NVM update steps - seamless recovery is a possibility if the NVM update gets interrupted.
- IAFW informs TXE to update SVN of discrete signed FW components of new IFWI.
- IAFW deletes the IFWI file from ESP(EFI System Partition) and also updates the FWUPDATE result to OS.
- IAFW clears the NVM UPDATE/FW UPDATE tracking UMIP variable
- IAFW performs a reboot to boot with new FW, enables WRITE PROTECT on Boot Partition.
- OS Capsule Update handler digesting the update status.

Figure 24-9. IA FW Workflow for Firmware Update




24.6.1 OS and EFI Communication During OTA

24.6.1.1 OTA Capsule Delivery to UEFI Context

There are two options for the OS to push the Capsule image to EFI context are listed below and with both the options FW update flow is unified, update happens during the next reboot phase:

1. **Via UpdateCapsule () UEFI API:** OS Auto Updater invokes UEFI Capsule API.
 - API is called in the Boot service context.
 - EFI Capsule API creates a file to store the Capsule on EFI system partition [ESP] @ root directory.
2. **Capsule on disk approach-as a file on ESP:** OS/EFI App firmware update tool will mount the partition and copy the file on ESP
 - Copy the IFWI image to [ESP] root directory.

24.6.1.2 Trigger for IFWI Update

On system Cold reboot/Boot up flow, EFI will perform the below check to trigger the IFWI update.

- IFWI presence on [ESP] root directory.

24.6.1.3 EFI Communicating the IFWI Update Result

Windows* uses ESRT Table defined in Windows* UEFI Firmware Update Spec

1. ESRT is catalog [Table] in DRAM – POINTER communicated through EFI System Table Windows* expects this pointer through EFI System table.
2. Carries the Updatable firmware entities details like with a GUID/Version info/Result etc
3. UEFI 2.4 Spec – ESRT is not available for public usage and implementation of ESRT would be similar to BYT/CHT platform.

Android* to make use of UEFI 2.4 spec defined EFI_CAPSULE_REPORT Structure to get the Result Variable.

1. This structure is defined in the EFI 2.4 Spec section 7.5.6 and refers to that section for more details on the structure.
2. BIOS has to report the IFWI update result as part of the 'CapsuleStatus' member of EFI_CAPSULE_REPORT structure. Refer EFI 2.4 spec section 7.5.6 on error code details.



24.6.2 Pre Update Check Requirements

Power Check:

- System must have at least 25% battery charge – this check is not necessary for DnX based recovery.
- Tethered power (power via USB cable and/or AC power) is not required.

Security Check:

This check is composed of 3 sub checks:

- **Integrity Check:** Perform an integrity check on the firmware update payload.
- **Version Check:** Verify that the firmware being applied does not downgrade the current installed firmware, beyond the LowestSupported FirmwareVersion value.
- **Sign verification** of the capsule driver that updates the FW.
- **NIST** recommended firmware protection guidelines (NIST publication 800-147) must be adhered
 - System boot firmware must be protected.
 - System boot firmware updates must be signed.
 - System boot firmware protection cannot be bypassed.
 - A user must be present for all ‘System boot firmware’ updates.
- There shall be anti-rollback protection.

Storage Check:

This is done as appropriate depending upon the system’s hardware

- There is sufficient room for backups of the current firmware which will be replaced
- There is sufficient room in the device to accommodate the new firmware.

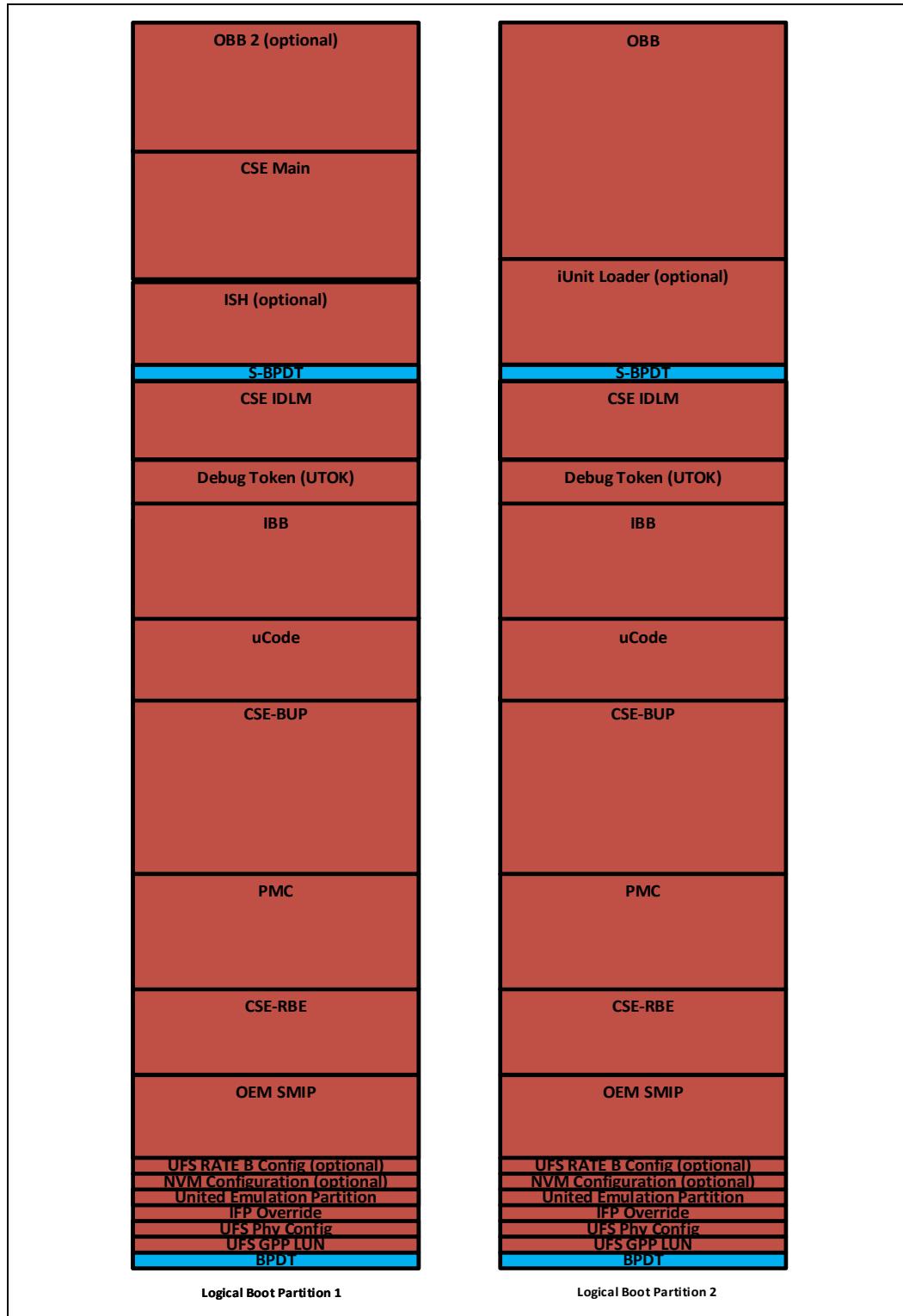
24.6.3 Post Update Check Requirements

- Any failure must result in an appropriate Last Attempt Status error code to OS via ESRT [A table in memory].
- Auth fail/version check fail /Battery check fail—must reject firmware and provide reasons to OS.
- Recovering from update Failures:
 - Retry the update 3 times to recovery from failed updates – forced shutdown case by Battery removal or device error.

24.7 Firmware Capsule Update Image Layout on Platform

For APL, irrespective of the firmware boot media, the firmware image layout is kept common other than a few low-level device-specific details. Supported media types are eMMC, UFS and SPI (only for APL). Layout is common between Android* and Windows* platforms. The system firmware layout on eMMC, UFS and SPI (only for APL) flash is as shown in figure below.

Figure 24-10. System Firmware Layout

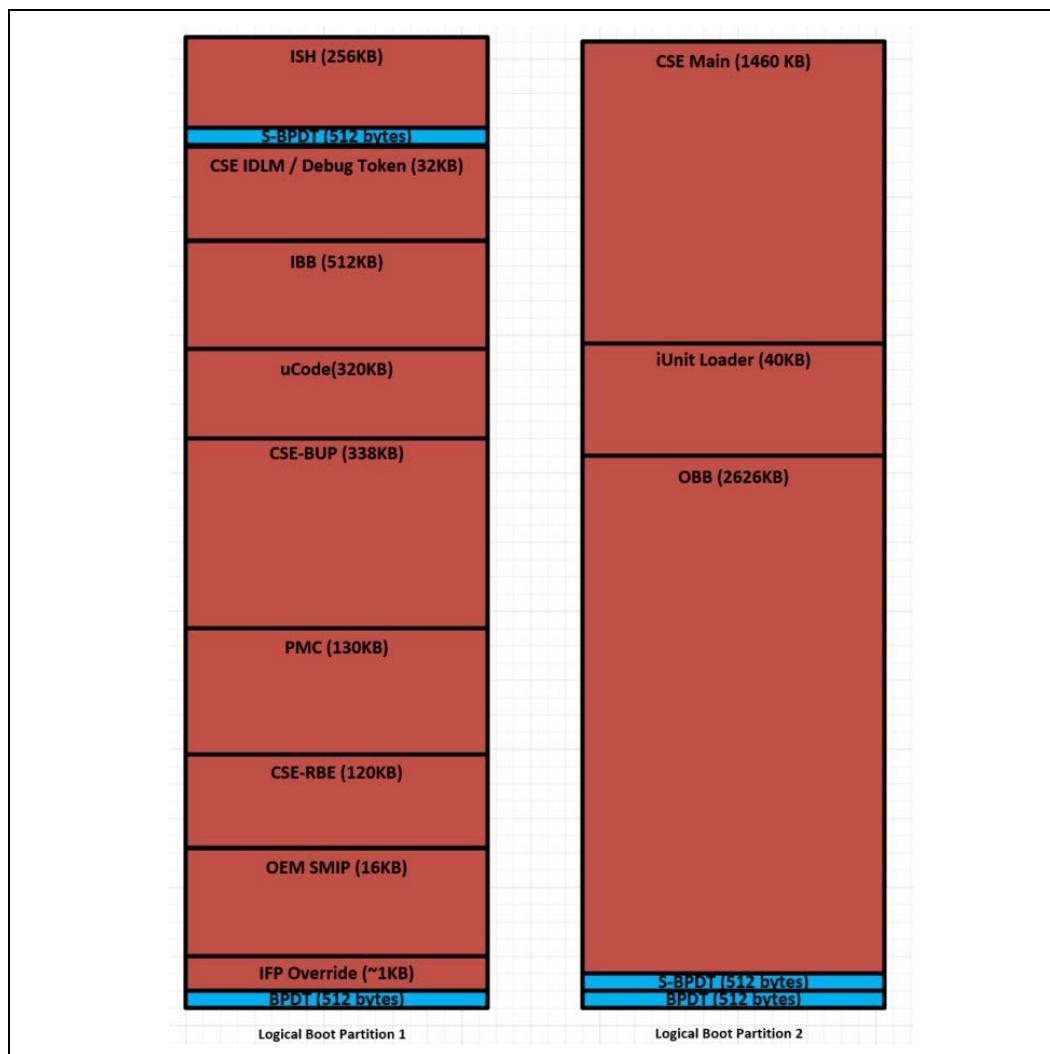


The two logical boot partitions contains the IFWI firmware, and refers to BP1+BP2 (respectively) in an eMMC/UFS device, or the first and seconds halves of the IAFW/TXE device region in a SPI device.

Regardless of the media type, the device area allocated for IFWI is split into two equal halves, called "Logical Boot Partition 1" (LBP1) and "Logical Boot Partition 2" (LBP2). At the base of each such LBP there is the Boot Partition Descriptor Table (BPDT) which defines logical sub-partitions in that LBP, composed of a header and a variable number of entries. The components that are critical for boot are called "critical sub-partitions" and are the first components in the flash. They might also be duplicated on both LBPs, as depicted in the image above. Following the critical sub-partitions is the Secondary BPDT, which points to the non-critical sub-partitions in the LBP.

The layout also supports the option where the critical sub-partitions are not redundant, as seen in figure below.

Figure 24-11. Image Layout with Critical Sub-Section to Redundant





Note: The recommended size for SPI NOR flash is 8MB at least and the sizes depicted in the diagrams are maximum sizes for 8MB SPI NOR. The logical sub-partitions may be smaller. There are no alignment requirements for logical partition start offset or for logical partition sizes.

The **BPDT** splits the LBP into logical sub-partitions. It is limited to 4KB (although in reality is much smaller). The BPDT structure is further defined in the next section.

The **S-BPDT** is a secondary BPDT, of the same structure as the BPDT, defining non-critical sub-partition.

The **UFS Phy Config** sub-partition contains configuration values for the UFS phy, consumed by ROM to calibrate the UFS phy for high speed device access in case UFS is used. The content of this sub-partition is provided by the UFS phy team and the OEM tools do not provide means to override it. Refer the ROM FAS for details.

The **UFS GPP LUN ID** sub-partition contains the LUN ID for the general purpose partition on the GPP for the TXE use, as configured by the OEM. It consists of the LUN ID, and a 2 byte CRC.

The **IFP Override** sub-partition contains overrides for IFPs, used by ROM/RBE in manufacturing mode. Refer the ROM FAS and RBE FAS for details.

The **OEM SMIP** sub-partition (**SMIP = Signed Master Image Profile**) contains OEM-signed configuration parameters for the platform. The sub-partition contains the following:

- A directory
- A partition manifest
- An SMIP structure, signed by the manifest. The SMIP structure is defined in details in the SMIP section.

The **TXE-RBE** sub-partition contains the TXE initial boot code. It contains the following:

- A directory
- A partition manifest, signed by the TXE ROM key
- The TXE RBE (ROM Boot Extension) module code

The **PMC** sub-partition contains the PMC firmware. It contains the following:

- A directory
- A partition manifest, signed by a key in the RoT key manifest
- 2 PMC firmware code modules for 2 different hardware SKUs
- A PMC configuration block (Intel PMC SMIP; this is in addition to the PMC OEM SMIP that is present in the SMIP sub-partition)

The **TXE-BUP** sub-partition contains the TXE second stage boot code. It contains the following:

- A directory
- A RoT key manifest
- An OEM key manifest



- A partition manifest
- TXE Kernel
- TXE Bringup process
- TXE System Library
- TXE Intel data – default values, a.k.a. intel.cfg
 - OEM Intel data (not signed) – OEM override values, a.k.a. fit.cfg

Note: TXE Kernel, TXE Bringup and TXE System Library are **paged components**, i.e. they are brought page-by-page on demand from the NVM device by the kernel using the device DMA. Therefore, for performance reasons, they must be **4KB aligned** on the NVM device.

The **uCode** sub-partition contains CPU microcode patches. It contains the following:

- A directory
- Two unified (CPU + Punit) patches, for 2 different hardware SKUs

Note: The patches are signed by the CPU and therefore an external manifest is not required

The **IBB** sub-partition contains the critical IAFW code. It contains the following:

- A directory
- A partition manifest. This manifest also contains the hash for the OBB.
- IBB-L module
- IBB module

The **OB** sub-partition contains the rest of the IAFW code. It contains the following:

- A directory
- One or more OBB modules as designed by the OEM (loader, main, recovery and so on.)

The **TXE-Main** sub-partition contains the main TXE code. It contains the following:

- A directory
- A partition manifest
- A list of TXE modules

The **iUnit** sub-partition contains the boot loader for the iUnit engine. It contains the following:

- A directory
- A partition manifest
- The iUnit ROM-replacement loader code, as loaded by TXE

The **ISH** sub-partition contains the ISH code. It contains the following:

- A directory
- A partition manifest
- The ISH code binary



The **IDLM** sub-partition is a placeholder (not built into the default image) for a TXE debug module that can be signed by Intel for the OEM debugging facilities. It contains the following:

- A directory
- A partition manifest, signed by the TXE ROM key
- The TXE IDLM module code

24.7.1 Boot Partition Descriptor Table

The Boot Partition Descriptor Table (BPDT) is a table of offsets to all individual sub-partitions contained within the LBP. A sub-partition is defined as a signed data block with the manifest available at the beginning allowing authentication.

Each LBP contains two BPDT structures: the main BPDT at offset 0 of the LBP, which points to the critical sub-partitions and to the secondary BPDT; and a secondary BPDT in another location in the partition, pointing to the non-critical sub-partitions.

The BPDT contains a header, immediately followed by 0 or more entries (number of following entries is indicated in the header).

Note: BPDT is not signed and therefore its consumers must treat its contents with care.

BPDT header:

Name	Offset	Size (bytes)	Description
Signature	0	4	Validity signature. For a valid BPDT (aka "green"), this value must be 0x000055AA. During IFWI update, this value is modified. The value of 0x00AA55AA indicates the BPDT is valid and can be booted from, however the firmware update is still in progress (aka "yellow" - recovery mode). Any other value indicates an invalid BPDT structure (aka "red").
Descriptor Count	4	2	Number of BPDT entries following this header
Version	6	2	Version of this BPDT structure. Must be 1 for APL
Redundant block XOR SUM	8	4	The XOR Checksum of the redundant block (from the beginning of the BPDT structure, up to and including the S-BPDT), i.e. such that the XOR checksum of the redundant block <i>including</i> this field is 0. If no redundancy supported, this field is marked as 0.
IFWI Version	12	4	Version of the particular IFWI build as marked by the build server
Reserved	16	8	Must be 0



BPDT Entry

Name	Offset	Size (bytes)	Description
Type	0	4	<p>Bits 0:15 - type of the logical sub-partition indicated by this entry. Should be one of the following:</p> <p>0 = OEM SMIP 1 = TXE RBE 2 = TXE BUP 3 = uCode 4 = IBB 5 = S-BPDT 6 = OBB 7 = TXE Main 8 = ISH 9 = TXE IDLM 10 = IFP Override 11 = Debug Tokens 12 = UFS Phy Configuration 13 = UFS GPP LUN ID 14 = PMC 15 = iUnit</p> <p>Bits 16:31 – flags. A bitmask indicating special attributes for the sub-partition indicated by this entry. Currently the following flags are defined:</p> <p>Bit 16: split sub-partition, first part: when this bit is set, the entry indicates a split sub-partition, where the first part of it is located in the LBP containing this entry and second part is located in the other LBP. May be set for up to one sub-partition only. Applicable for non-critical sub-partitions only (may appear only in S-BPDT).</p> <p>NOTE: This split can only be used for OBB sub-partition.</p> <p>Bit 17: split sub-partition, second part: when this bit is set, the entry indicates a split sub-partition, where the first part of it is located in the other LBP and second part is located in the LBP containing this entry. May be set for up to one sub-partition only. Applicable for non-critical sub-partitions only (may appear only in S-BPDT).</p> <p>NOTE: This split can only be used for OBB sub-partition.</p> <p>Bit 18: sub-partition contains directory structure</p>



Name	Offset	Size (bytes)	Description
			Bit 19: sub-partition copied to DRAM cache: indicates whether the sub-partition is copied to DRAM cache, i.e. whether it is required for a reset flow. All sub-partition with the exception of BIOS sub-partitions should have this bit set.
Sub-partition offset	4	4	Offset of the logical sub-partition indicated by this entry. The offset is indicated in bytes from the beginning of the LBP.
Sub-partition size	8	4	Size of the logical sub-partition indicated by this entry. The size is indicated in bytes.

Note: In order to maintain a simplified TXE ROM and RBE code, the following order is enforced in the first items in the BPDT:

- 1) TXE-IDLM
- 2) IFP Override
- 3) S-BPDT
- 4) TXE-RBE
- 5) UFS Phy Config
- 6) TXE-BUP

There is no order guarantee for any of the other entries in the BPDT.

24.7.2 Logical Sub-Partition Layout

24.7.2.1 Sub-Partition Directory

At the base of each logical sub-partition there is a sub-partition directory, composed of a header and a variable number of entries. The sub-partition directory points to the "files" in the sub-partition – process code binaries, manifests and lookup tables. The sub-partition directory is not signed.

24.7.2.2 Sub-Partition Directory Header

Name	Offset	Size (bytes)	Description
Header Marker	0	4	Header marker, must be 0x44504324 ("\$CPD") for this version
Number of Entries	4	4	Number of entries following the header
Header Version	8	1	Must be 1 for this version
Entry Version	9	1	Must be 1 for this version
Header Length	10	1	In bytes, equals 0x10 (16) for this version



Name	Offset	Size (bytes)	Description
Checksum	11	1	8-bit XOR Checksum of the sub-partition directory (from 1 st byte of header to last byte of last "partition directory entry")
Sub-Partition Name	12	4	ASCII short name for the partition

24.7.3 Sub-Partition Directory Entry

Name	Offset	Size (bytes)	Description
Entry Name	0	12	Character string; if the name is shorter than the field size, the name is padded with 0 bytes.
Offset	12	4	Bits 0:24 – offset – This is the offset from the <i>beginning of the logical sub-partition</i> .
Reserved			Bit 25 – Huffman compressed Y/N (<i>placeholder – Huffman compression not supported in APL</i>) Bits 26:31 – reserved, set to 0
Length	16	4	Module length, in bytes. For Huffman-compressed modules, this refers to the uncompressed size. For software-compressed modules, this refers to the compressed size.
Reserved	20	4	Must be 0

24.7.4 SMIP

The **Signed Master Image Profile (SMIP)** contains platform-specific data that firmware and software may find necessary in generating specific platform behavior. Currently, only an OEM-signed SMIP is in use.

The SMIP is required to begin with a SMIP Descriptor Table (SDT) that helps locate the remaining blocks within the SMIP. Required blocks in SMIP are those dedicated for TXE, PMC, IAFW respectively in that order. SDT structure is defined below.

Name	Offset	Size (bytes)	Description
Number of Descriptors	0	2	Number of SMIP blocks ('n') inside this SMIP structure
Size of SMIP	2	2	Size, in bytes, of this SMIP structure (including the SDT structure)
Block 0 Type	4	2	Type of block 0. Can be one of the following: 0 = TXE 1 = PMC



Name	Offset	Size (bytes)	Description
			2 = IAFW
Block 0 Offset	6	2	Offset of block 0
Block 0 Length	8	2	Length of block 0 in bytes
Block 0 Reserved	10	2	Must be 0
Block 1 Type	12	2	
Block 1 Offset	14	2	
Block 1 Length	16	2	Length of block 0 in bytes
Block 1 Reserved	18	2	Must be 0
....			
Block 'n-1' Type		2	
Block 'n-1' Offset		2	
Block 0 Length		2	
Block 0 Reserved		2	

24.7.5 Manifest

Most sub-partitions contain a partition manifest file. The manifest is signed by a key, which could be signed by one of the ROM hardcoded keys, or by a key in one of the key manifests found in the TXE-BUP partitions. The manifest file may contain data on the system policies, on the sub-partition, and/or on the binaries (processes / shared libraries / data files) that are included in the sub-partition.

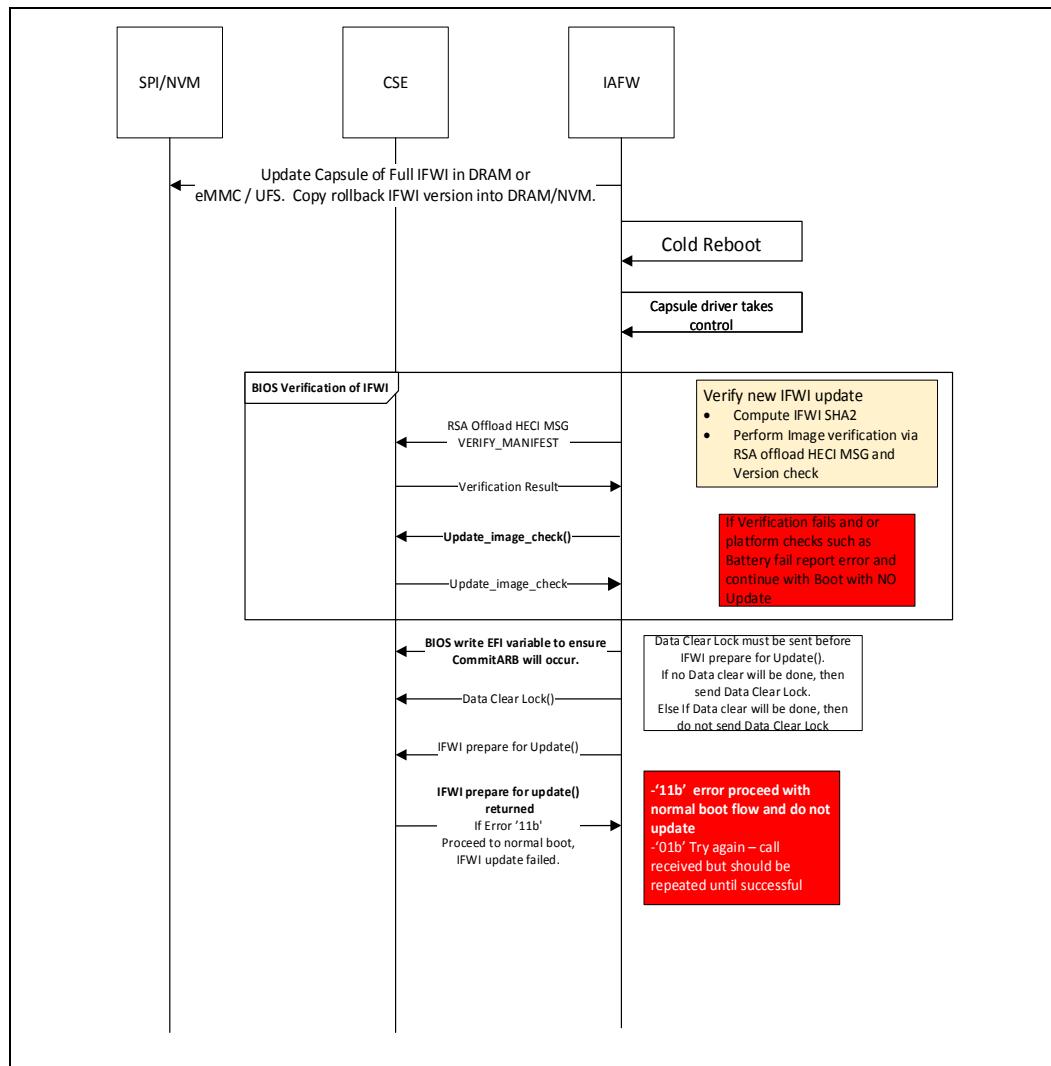
24.8 Overall IFWI Update Sequence

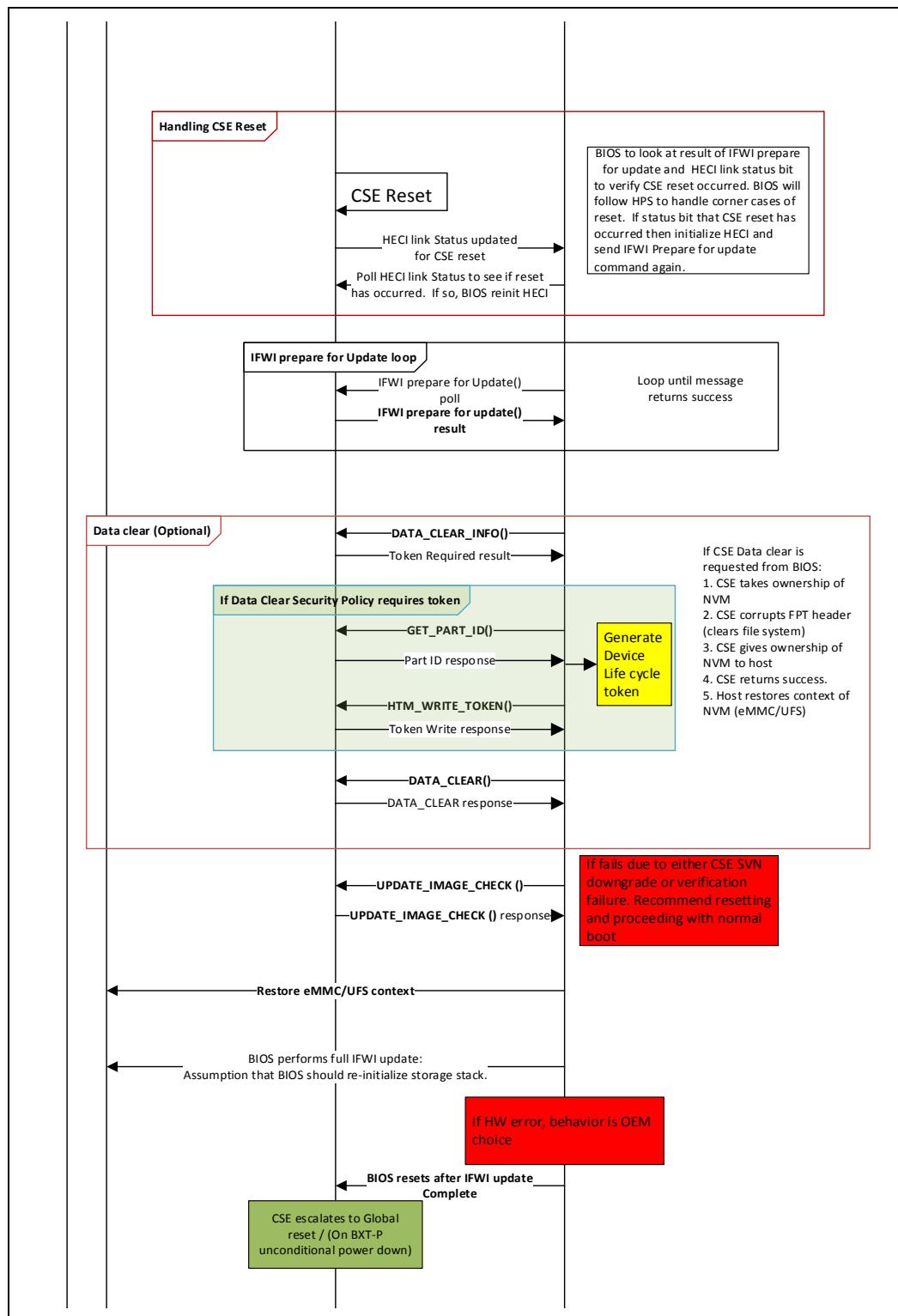
Platform Level Flow

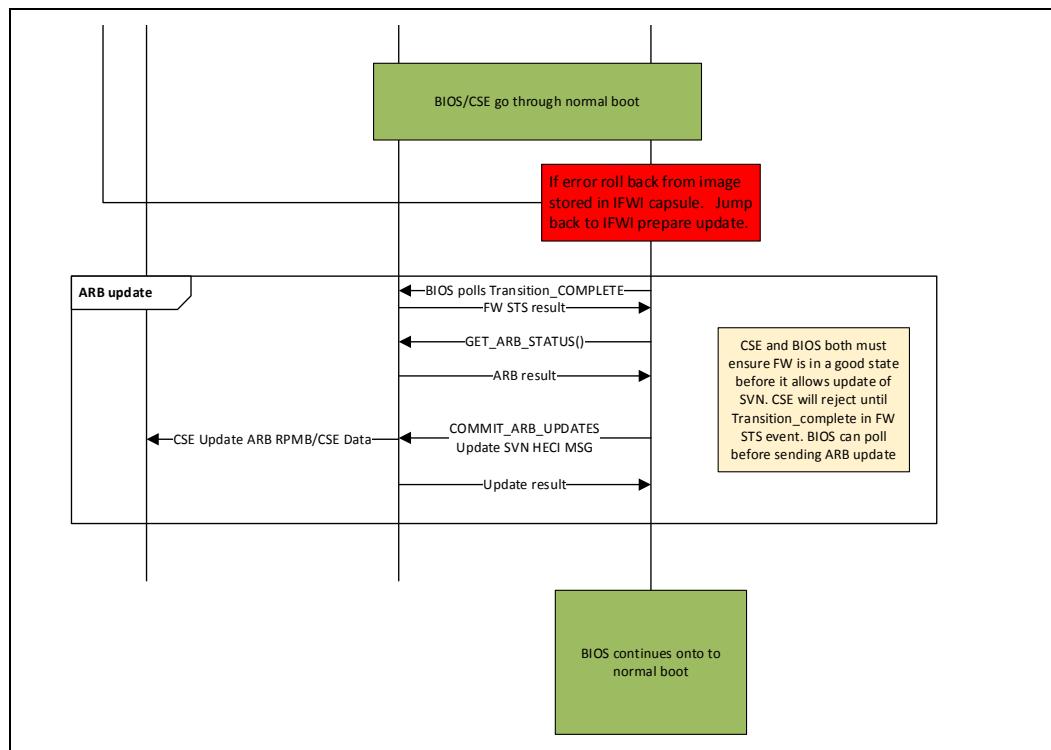
1. OS passing FW image to EFI [IAFW] context (ESP - Capsule in DRAM or NVM)
2. System reboot
3. IAFW (from Capsule driver) takes control
4. IAFW performing IFWI image authentication (VERIFY_MANIFEST()) and IFWI version check
5. IAFW performing battery health check and timer bound PBTN disable [optional]
6. IAFW sends IFWI prepare for update() to quiesce CSE FW write activity
 - a. IAFW repeats IFWI prepare for update command until successful result
7. IAFW perform checks UPDATE_IMAGE_CHECK to ensure no SVN down grades happening as well as prepare for CSE SVN upgrade.
8. IAFW performs fault tolerant update NVM update steps - seamless recovery is a possibility if the NVM update gets interrupted
9. IAFW performs a reboot to boot with new FW

- a. IAFW does cold reset, CSE escalates to unconditional power down or GRST. (GRST on APL does not clear all context in all cases)
- 10. IAFW informs CSE to update SVN of discrete signed FW components of new IFWI
 - a. If IAFW fails SVN check or does not properly load with new FW loaded on step 8, IAFW will reboot and attempt to roll back to original FW that is in Capsule.
- 11. IAFW deleting the IFWI file from ESP (DRAM/NVM capsule) and also updating the FWUPDATE result to OS
- 12. IAFW clearing the NVM UPDATE/FW UPDATE tracking variable
- 13. IAFW Enables WRITE PROTECT on Boot Partition

Figure 24-12. IFWI Prepare for Update Flow









24.8.1 Detailed IFWI Prepare for Update Details

IFWI Prepare for Update will quiesce CSE activity on NVM and give Host ownership of the NVM. IAFW will send this command in the overall flow described [Overall IFWI Update Sequence](#). The IFWI prepare for update is a command that needs to be polled for a successful result. IAFW/BIOS must be prepared for a CSE FW update to happen during this flow and must re-instantiate HECL when the link goes down after a reset. To minimize polling it is recommended to wait until the reset in the non-recovery flow.

Note: For BIOS this could be a long flow it can take as long as 1 min under worst case scenario. If CSE FW cannot support FW update through regular means, would recommend at least one global reset to try again. BIOS can blindly update. Recommend to have a bread crumb or track to ensure no perpetual loop.

1. **(Optional) BIOS sends Update Image Check API**
 - a. Image will return pass or fail (fail meaning that it does not pass security checks)
2. **Data clear/Data Clear lock:Refer [Data Clear](#) for more details**
 - a. If not performing Data clear then BIOS/IAFW must send DATA clear Lock(). Data Clear Lock API MUST be sent before IFWI prepare for update.
 - b. If performing data clear do not send Data clear Lock()
 - i. If data clear is required in and <non recovery mode> Data clear requires IFWI prepare for update to be successful in order to run.
 - ii. If data clear is required and platform is in <recovery mode> IFWI prepare for update should still be run. This will effectively start at [step 4.b](#)
3. **Host send IFWI prepare for update:** If CSE FW initialization is not complete (either in normal or recovery mode or FPO), return error (Try Again message).
 - a. Note: that if only BUP is loaded in recovery mode, then being in BUP in recovery mode means FW initialization is done. Start at [4.b](#)
 - b. POST EOM: If command sent prior to DID or after EOP, return Availability error. Post EOP Host should not be able to execute this command. Post DID and pre- End of POST (EOP) is a pre-requisite for instantiating this command.
 - c. PRE EOM: command needs to be sent after DID. Command can be successful after EOP as long as End of Manufacturing is not set.
4. **Branch according to whether handling is in policy:**
 - a. If handling in Policy:
 - i. Mark an AON SRAM breadcrumb called PREPARE_FOR_UPDATE about need to enter "IFWI update ready" mode.
 - ii. Perform CSE FW reset.
 - iii. Next CSE boot:
 - RBE identifies NOT to escalate to platform reset.
 - BUP identifies that IFWI update was requested (from bread crumb). Issues BUP_CSE_RECOVERY event, and does not proceed loading rest of FW.
 - BUP set HECL Link bit that CSE Rest has successfully occurred.
 - iv. Both before reset and after reset, whenever the "prepare for IFWI update" is received, the "**Try Again**" value is returned until **step 8**.
 - b. Else if in Bring Up (If called from CSE Recovery Mode):



- i. CSE independently identified recovery mode or FPO – issue BUP_CSE_RECOVERY event.
 - ii. Whenever the “prepare for IFWI update” is received, the “**Try Again**” value is returned until **step 8**
 - iii. If BUP image source == DRAM (booted from cache)
 - If yes
 - a. Set both AON bread crumb IGNORE_PLATFORM_BOOT_CRITICAL_SECTIONS to ‘11’ to ensure ME reset is not escalated.
 - b. Take CSE ownership of NVM.
 - c. Clear AON SRAM bit to prevent going to DRAM cache: “**NVM Access Prohibited**”
 - d. Issue CSE reset
 - Else continue on to **next step** without reset
 - c. Else if in Bring Up (if NOT called from Recovery)
 - i. Bring up terminates polling on HECI
 - ii. Policy will handle continue flow from on step 4.a
5. At this point, we are in BUP, after it issued BUP_CSE_RECOVERY event. Components perform the following handling of this event:
 - a. BUP HECI: When BUP is in recovery mode it must continue to own HECI 1. (Must not postpone handling until CSE Main)
 - b. All components: Read any data from NVM to SRAM which might be needed in steps below, because data access might be lost (e.g in case of data clear or FPO). NOTE: **This will happen on event entry**
 - i. As FW initialization is complete, this halts all CSME accesses to the IFWI region of NVM and now BIOS may update any portion of the IFWI region of NVM. Page access can no longer access NVM.
 - c. BUP Storage: For relevant NVM devices (e.g eMMC, UFS), move ownership to host. **On event exit**
 - i. Any NVM access after this point must assert.
 6. When all BUP components have completed processing BUP_EV_CSE_RECOVERY event, BUP Maestro issues BUP_EV_CSE_DISABLED event.
 7. At this point, CSE is not performing any independent accesses to NVM. Still, it is possible that certain BIOS calls below will cause CSE to perform momentary accesses, but this will be done in a synchronous manner only.
 8. CSE returns **SUCCESS** on next “Prepare for IFWI Update” call.

Note: Storage must return this only after receiving BUP_EV_CSE_DISABLED event; this indicates that all components have processed BUP_EV_CSE_RECOVERY event.

a. **BIOS will need restore/re-initialize eMMC/UFS context**

Note: This only needs to done before Host writes to NVM. Data clear, also cause a NVM ownership handover

9. BUP is polling for HECI and PM as expected in recovery flow.

10. (Optional) In case BIOS/Host requests NVM data clear Data_clear() :

Note: This command requires a successful IFWI prepare for update to run.



- a. In case of relevant NVM devices (e.g eMMC, UFS), CSE takes over ownership.
- b. CSE performs data clear.
 - i. Note: Corruption of FPT header marker is enough for this.
- c. In case of relevant NVM devices (e.g eMMC, UFS), CSE relinquishes ownership to host.
- d. CSE returns success.
- e. It is BIOS's responsibility not to perform writes to NVM until NVM data clear returns SUCCESS.
- f. **BIOS will need restore/re-initialize eMMC/UFS context before host can write to NVM**
- g. Note: If this command requires a successful IFWI prepare for update to run.

11. BIOS restore/re-initialize eMMC/UFS context before host can write to NVM.

Note: This should be done before Host write to NVM after the IFWI prepare for update returns success.

- 12. BIOS/IAFW now performs the full FW update using direct write**
- 13. BIOS performs graceful cold reset**
 - a. BUP Maestro initiates relevant transition that was supplied in the IFWI Prepare for update command.
- 14. BIOS/IAFW should perform the following checks on every boot**
 - a. Both BPDT1 and BPDT2 must both be green, or will need to recover/complete update or roll back.

Note: There is a softstrap option where missing BPDT2 will not cause BUP to enter recovery

- b. Ensure CSE is still not in recovery mode
- c. If booting on platform with redundant critical sections use MPB – NVM to see if booting off of BPDT2. This tells BIOS/IAFW that there was corruption in the critical section on boot partition 1. Would recommend either copying boot partition 1's critical section to boot partition 2, or roll back to ensure we properly write the flash.
- 15. Upon completion of the update on next normal resume flow, BIOS to send command GET_ARB_STATUS.**
 - a. This will be rejected until transition complete. This is located in FW STS Register.
- 16. BIOS to send CommitARBUpdate() to update SVN. The SVNs from the new updated image will now be stored in the CSE data region.**
 - a. This will be rejected until transition complete. This is located in FW STS Register.
 - b. If error occurs here, recommendation is to roll back.

24.8.2 Detailed IFWI Prepare for Update Details (Recovery Flow/Flash Descriptor Override (FDO) strap behaviour)

CSE Recovery flow is incorporated into the above flow start at step [4.b](#)

If Flash descriptor override strap is asserted during boot, the platform will start off at step [4.b](#). If host begins to write before CSE has been quiesced, unintended behavior can happen as CSE in DRAM maybe out of sync with CSE in flash. It is recommended to have a successful IFWI prepare for update complete before writing to IFWI/CSE data region or flash descriptor.



Note: When FDO is asserted there are no FW interactions beyond BUP steps to prepare for update (CSE quieced and no longer accessing NVM.)

24.9 IFWI Recovery/Update Module

System firmware update might fail in any step listed in figure below. The failure will be recorded in ESRT table. Unlike device firmware update, IA FW should take necessary steps to recover from the failure before handing over to Windows* boot loader.

Fault tolerant update process ensures IA FW boot succeeds even after a failed update; while seamless recovery flow restores image layout on eMMC/UFS/SPI flash to normal state.

IBB should have the recovery support to trigger the IFWI update or enter the seamless recovery flow.

IBB will have the recovery support to trigger the IFWI update or enter the seamless recovery flow. IBB would verify the respective UEFI Variable and look for the IFWI.BIN presence on the EFI system partition to trigger the IFWI update.

In case of the corrupted OBB, recovery module will trigger the manual recovery if FW capsule image is not present on ESP or would trigger a seamless recovery if the FW image is present on the ESP.

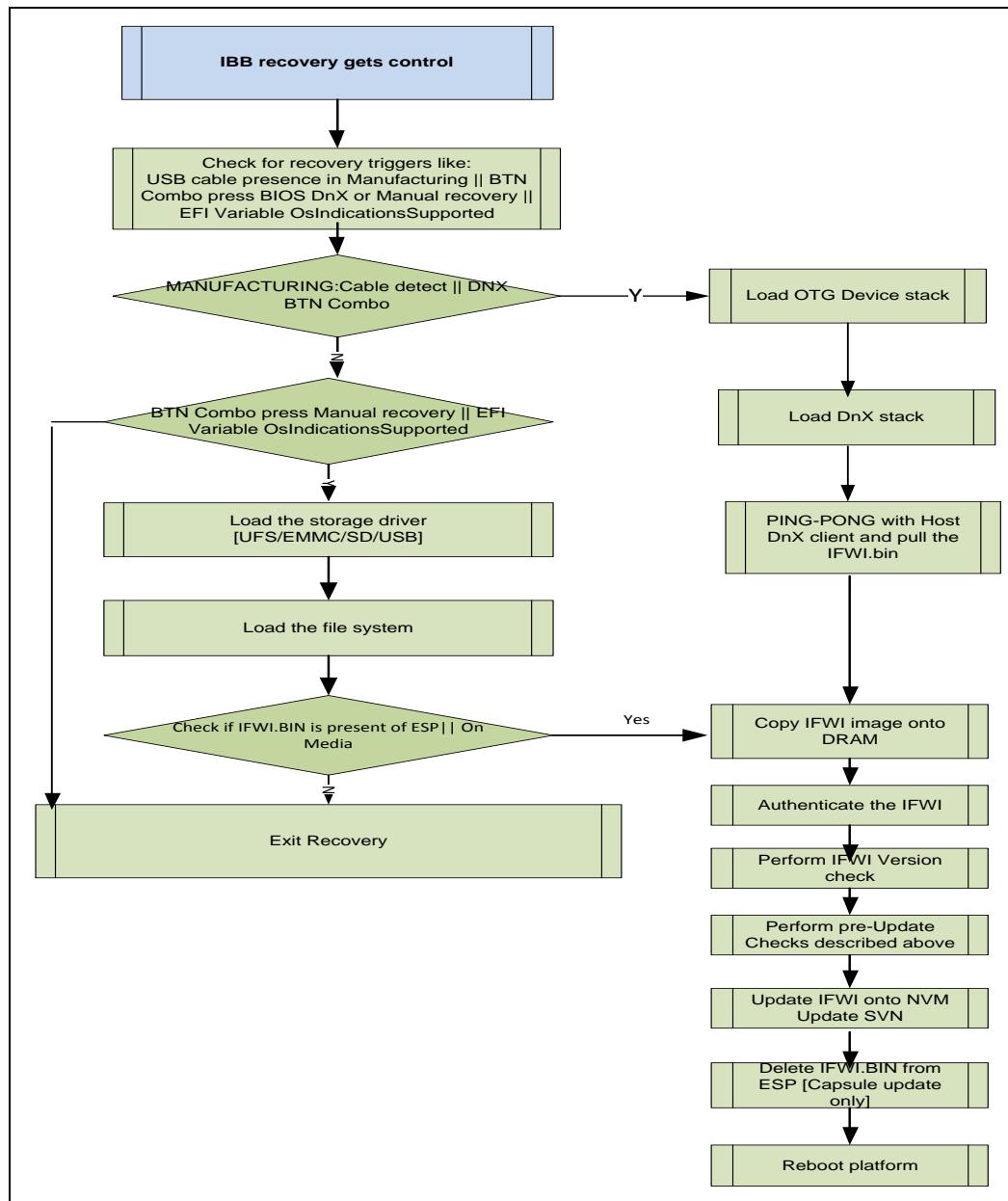
Boot after IFWI update (mainly with data clear) can cause a delay of 10-20sec in responses.

1st boot time can be a little slower by 5-10 sec.

24.9.1 Unified Recovery/Update Mechanism

Flow chart below outlines the IAFW recovery/Update flow. Recovery can be either DnX, or Manual via SD/USB which is optional or it could be via the Capsule update. All schemes are various means of getting the IFWI image into DRAM. Once that's done the common FW update module gets triggered to perform the FW image authentication/update.

Figure 24-13. Unified Recovery/Update Mechanism



24.9.3 Image Authentication

IFWI image as a whole needs to be signed and image needs to be subjected to signature/version checked before the FW update on to Flash boot partition. Benefits of sign verifying the IFWI as a whole is 2 fold:

- 1- Simplifies the image verification process, avoids discrete FW component verification; still maintenance the security.



- 2- Avoids mix and match of the discrete FW components that make the IFWI; good for security to avoid vulnerabilities caused by FW component mix and matches.

OEM key that was used to sign the OEM IAFW components is used to sign the IFWI. This key is part of SOC IFP FUSES. IAFW is expected to compute the SHA2 of the IFWI, offload the RSA verification of the IFWI to TXE by triggering HECI messages.

As part of this IPC, IAFW can also pass the new version info to TXE. TXE would cross-check the version info that's in the RPMB, to provide a green/no green on image verification/version check.

Note: This IFWI version that's on RPMB would only be used during the FW update. Also note that IAFW has not yet requested TXE to update the SVN of the discrete FW components.

Additional HECI message is required for IAFW to let know the TXE on upgrading the component SVN on RPMB. This message gets blocked out after EOP HECI message, thus BIOS can issue this HECI right after the FW update completion or on the IFWI image Logical view and update sequence.

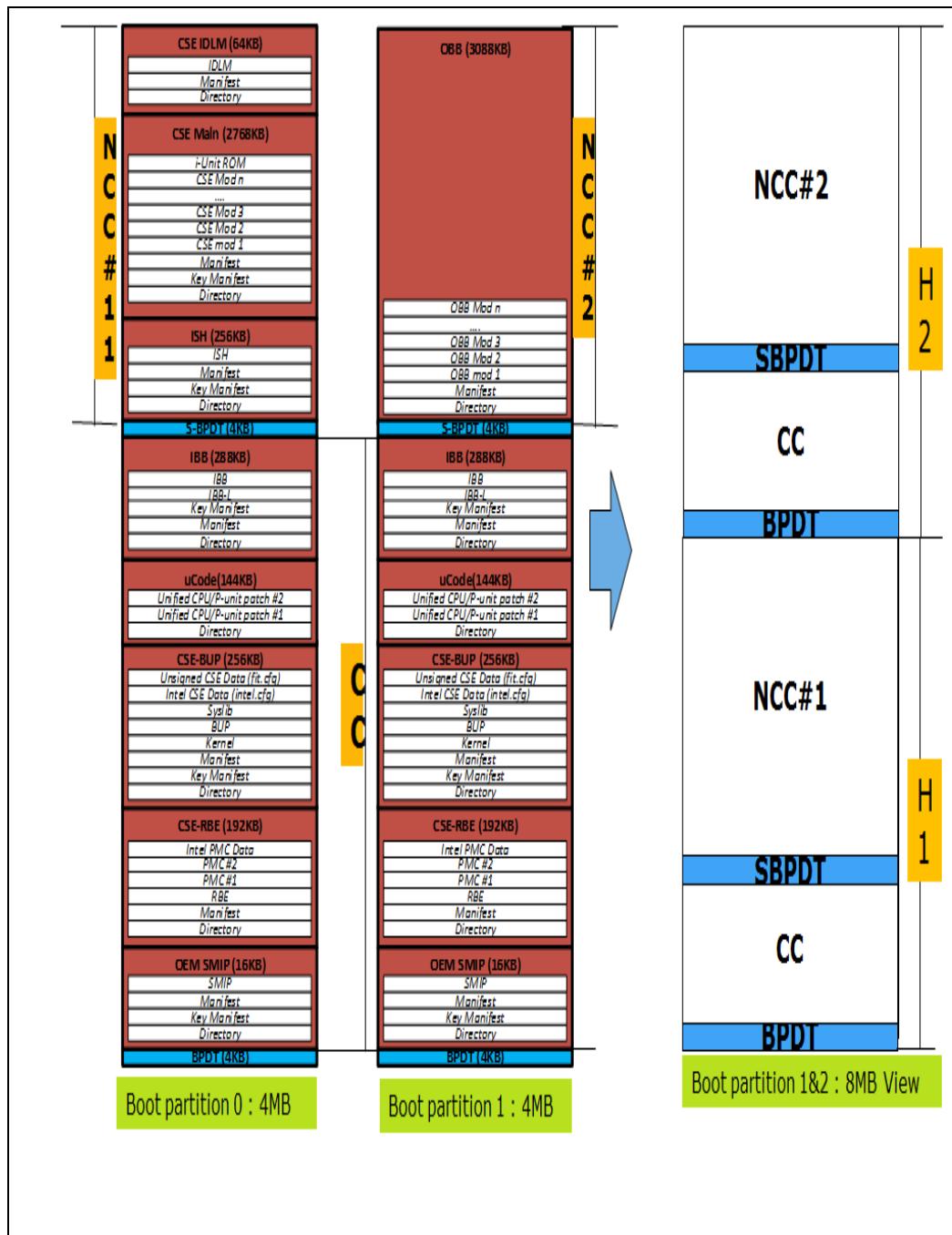
24.9.4 IFWI View from FW Update Perspective

APL support booting from eMMC/UFS devices and both these devices are having 2 4MB Boot partitions. As shown in the below diagram, there are 3 main regions of the IFWI.

- CC- Critical Components which is essential to boot IAFW which enables recovery; the list includes TXE RBE/TXE BUP/PMC patches/uCODE/SMIP/IBBL/IBB. CC can have an optional redundant copy as well.
- NCC-Non Critical Components which are essential for system boot but not needed for recovery. There are 2 NCC components.
 - NCC#1 – this region includes Intel FW components like TXE Main FW, ISH FW and so on.
 - NCC#2 –this region main includes OEM BIOS OBB/OBBX components

Below figure summarizes the APL IFWI image Layout on EMMC/UFS boot partition; both devices are recommended to have 2 x4MB Boot partition to hold Boot critical components and Non critical components. In order to explain the IFWI update steps, IFWI contents are termed as CC/NCC#1/NCC #2/BPDT/SBPDT.

Figure 24-14. IFWI View from IFWI Update Perspective



- BPDT at base of BP1 and BP2, with **two** validity markers.
 - Tri-state: invalid [red], valid [green] or semi-valid [yellow]
- BPDT points to secondary BPDT + critical components (CC)
 - IBB, TXE-FTP, PMC, UCODE, SMIP, UMIP and so on.
 - [BPDT **replaces and extends** the TXE FPT – single indirection to all partitions]
 - Addresses are relative to beginning of BPDT



- Secondary BPDT (SBPDT) points to non-critical components (NCC):
 - ISH, TXE-NFTP and so on.
- If redundancy enabled:
 - Both partitions have same BPDT, critical components duplicated.
- If no redundancy.
 - One of the BPDT's points to critical components. The other points to the secondary BPDT only.
- 8MB view – 2 4MB partitions are called as H1 and H2.
 - H1 is the partition that holds CC and Intel FW components like TXE/ISH and so on.
 - H2 is the partition that holds CC [optional] and OEM FW components like OBB
 - H1 and H2 may change their position at the end of every Firmware update sequence.
 - BIOS has to ensure it tracks which is H1 and H2 and then must follow the update sequence listed below.
 - IBB cannot assume OBB to be resident at a fixed offset of Boot partition – instead it has to read the BPDT and decide where to pick OBB or TXE has to indicate that to IBB.

24.9.5 Boot Partition Table Marking

- BPDT update signature can be of 3 types to indicate TXE if the partition is usable for system boot, used for recovery boot or not usable for boot.
- 3 types of BPDT signature schemes are represented in 2 color coding scheme to explain the state of the BPDT in the Boot partition 1[H1] and Boot Partition 2 [H2].
 - Green BPDT if it's with signature 0x54445042 ("BPDT"). TXE sees the Green BP only when the update is completed
 - Yellow BPDT: Yellow means this partition can be booted from, but we are still in recovery mode. In other words, if a reset occurs here, still need to go through recovery to complete the update. Otherwise if it remains green we might boot with one old partition and one new partition. If BPT signature is either 0x544450C2 ("bPDT") or 0x54445000 TXE consider it yellow (consider yellow #1 and yellow #2).
 - RED BPDT: BPDT with any other signature or FFFFFFFF is considered RED and TXE ignore such partitions for boot.

24.9.6 Detailed Fault Tolerant IFWI Update

Below listed sub sections outlines the IFWI update sequence i.e. 1st case addresses the IFWI update with redundant critical components and the 2nd case addresses the IFWI with no redundant critical component.

24.9.6.1 NVM Update sequence of Redundant Critical Component

This flow is used for eMMC/UFS/Block storage where there is duplicate critical components.

This section summarizes the sequence of IFWI NVM update that BIOS follows for a case with 'critical components' being replicated on both the boot partitions.



Update scheme that BIOS follows ensures fault tolerance by keeping the CC[Old] or CC[New] intact either on H1 or H2 partition and respective BPDT markers as Yellow. At any given time CSE ROM would get to pick CC region with 'Old CC content' or 'New CC content' to ensure IBB gets the control to enable seamless recovery to continue the interrupted FW UPDATE process.

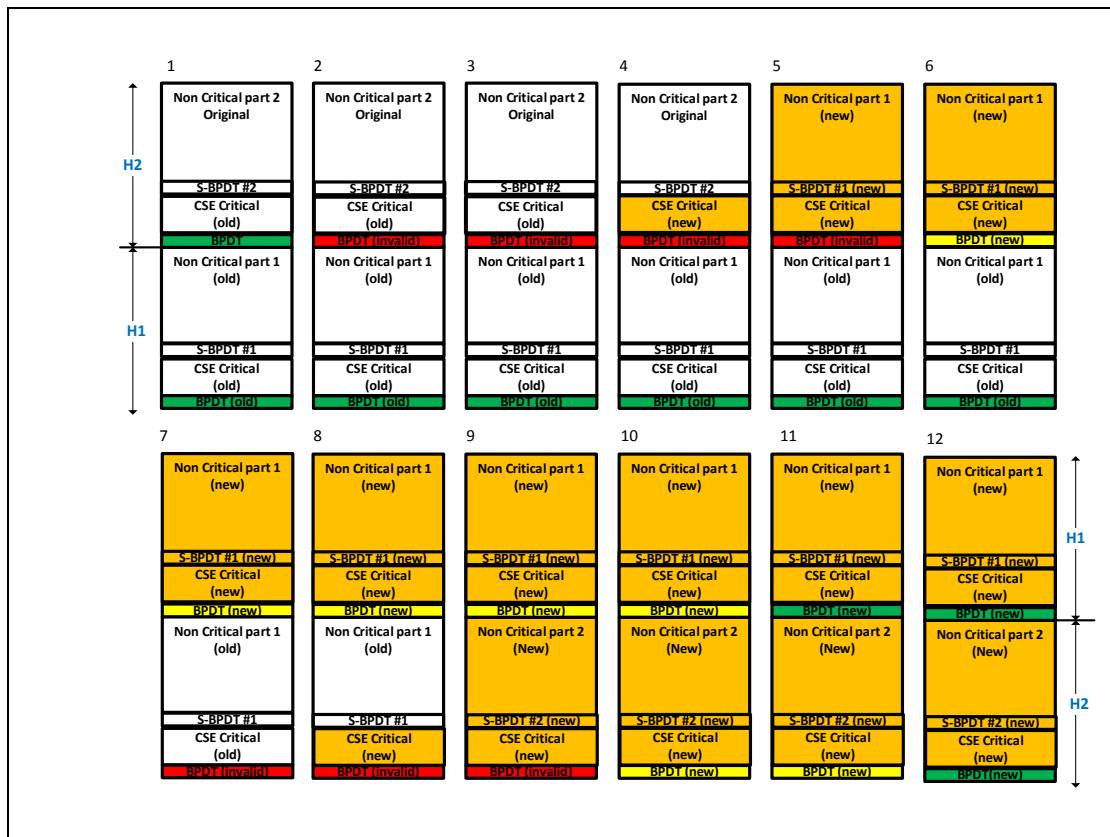
Note: Green to yellow transitions are not power fault tolerant as it requires an erase + write in order to be written. Due to this, any host/IAFW/BIOS update must look at both BPDT 1 and BPDT 2 and ensure both are green to know that IFWI is not in an update/recovery.

Flow assumes that BIOS has the FW Update/recovery sequence code in the IBB. IFWI NVM update sequence begins by BIOS tracking Boot partition H1 and H2 by reading the appropriate BPDT content. So the summarized NVM update sequence and the NVM update progress would be tracked on the UMIP FWUPDATE_PROGRESS field:

1. Begin with existing IFWI – BIOS to mark BP H1 and BP H2.
2. Invalidate H2 BPDT by updating its signature to 0xFFFFFFFF
3. Copy new Critical component [CC New] to right above H2 BPDT
4. Copy new Non critical Partition #1 [NCC #1] and SBPDT#1 On BP H2, right above CC New.
5. Update the new BPDT at H2 and make its marker Yellow
6. Invalidate H1 BPDT by updating its signature to 0xFFFFFFFF
7. Copy new Critical component [CC New] to right above original H1 BPDT
8. Copy new Non critical Partition #2 [NCC #2] and SBPDT#2 On BP H1, right above CC New.
9. Update the BPDT at H2 and make its marker Yellow.
10. Update the new BPDT at H1 and make its marker Green.
11. Update the new BPDT at H2 and make its marker Green.

Note: At the end of the IFWI update H1/H2 partition space is swapped.

Figure 24-15. Fault Tolerant Update with Duplicate Critical Partitions



24.9.6.2 NVM Update sequence for Non-redundant Critical Component (Without Swapping H1/H2)

This is the POR method of updating IFWI for APL.

There is a performance impact as there are extra flash writes necessary.

Note: Green to yellow transitions are not power fault tolerant as it requires an erase + write in order to be written. Due to this, any host/IAFW/BIOS update must look at both BPDT 1 and BPDT 2 and ensure both are green to know that IFWI is not in an update/recovery.

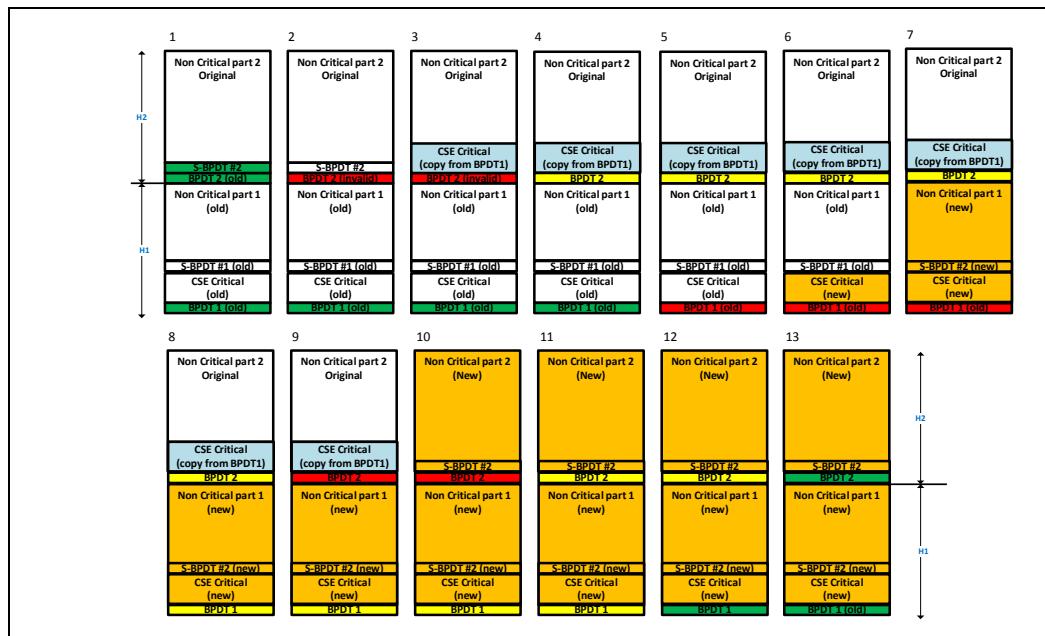
Note: There is a softstrap option where missing BPDT2 will not cause BUP to enter recovery.

1. Begin with existing IFWI – BIOS to mark BP H1 and BP H2 by tracking the 2 BPDT contents.
2. Invalidate H2 BPDT by updating its signature to OXFFFFFF.
3. Copy original Critical component from BPDT1 to right above H2 BPDT.
4. Update the BPDT2 at H2 and make its marker Yellow.
5. Invalidate H1 BPDT1 by updating its signature to OXFFFFFF.

6. Copy new Critical component [CC New] to right above H1 BPDT.
7. Copy new Non critical Partition #1 [NCC #1] and SBPDT#1 On BP H2, right above CC New.
8. Update the new BPDT1 at H1 and make its marker Yellow.
9. Invalidate H2 BPDT2 by updating its signature to OXFFFFFF.
10. Copy new Non critical Partition #2 [NCC #2] and SBPDT#2 On BP H2, right above original H2 BPDT2.
11. Update the new BPDT at H2 and make its marker Yellow.
12. Update the new BPDT at H1 to make its marker Green.
13. Update the new BPDT at H2 to make its marker Green.

Note: At the end of the IFWI update H1/H2 partition space are NOT swapped.

Figure 6 Fault Tolerant IFWI update single Critical Partition (APL-SPI only/ NO SWAP)



24.9.6.3 NVM Update Sequence for Non-redundant Critical Component (Not Used)

Included in the reference for SPI. This is a faster way to perform the update but is not currently POR for APL.

The sequence listed below is same as the previous update scheme with the exception of CC is being updated on only one BP.

At the end of the IFWI update Partition H1/H2 of OLD IFWI is swapped on new IFWI.

So the summarized NVM update sequence and of NVM update progress would be tracked on the UMIP FWUPDATE_PROGRESS field:

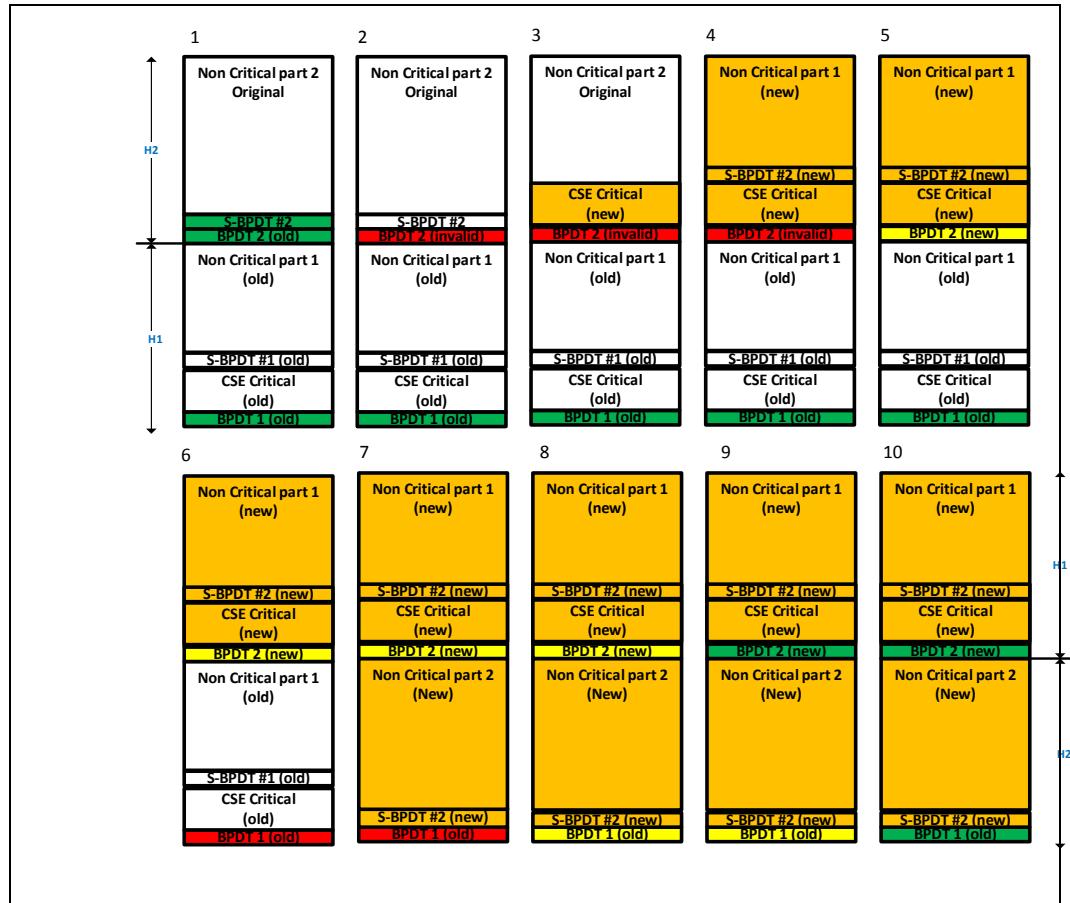


Note: Green to yellow transitions are not power fault tolerant as it requires an erase + write in order to be written. Due to this, any host/IAFW/BIOS update must look at both BPDT 1 and BPDT 2 and ensure both are green to know that IFWI is not in an update/recovery.

1. Begin with existing IFWI – BIOS to mark BP H1 and BP H2 by tracking the 2 BPDT contents.
2. Invalidate H2 BPDT by updating its signature to 0xFFFFFFFF.
3. Copy new Critical component [CC New] to right above H2 BPDT.
4. Copy new Non critical Partition #1 [NCC #1] and SBPDT#1 On BP H2, right above CC New.
5. Update the new BPDT at H2 and make its marker Yellow.
6. Invalidate H1 BPDT by updating its signature to 0xFFFFFFFF.
7. Copy new Non critical Partition #2 [NCC #2] and SBPDT#2 On BP H1, right above original H1 BPDT.
8. Update the original H1 BPDT to H2 BPDT and make its marker Yellow.
9. Update the new BPDT at H1 to make its marker Green.
10. Update the new BPDT at H2 to make its marker Green.

Note: At the end of the IFWI update, H1/H2 partition space is swapped.

Figure 24-16 Fault Tolerant IFWI Update Single Critical Partition (APL-SPI only)



24.9.7 Recovery Mechanism

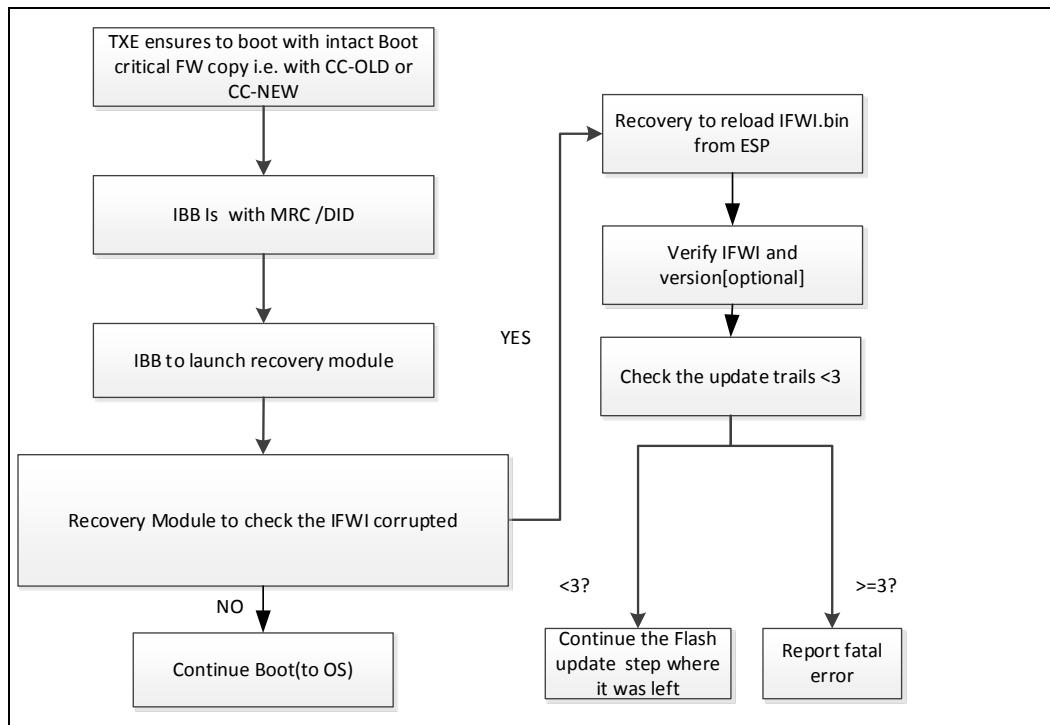
During firmware Update, failures chances are minimized by performing.

- Battery capacity is ensured – [5%+ but OEM choice].
- PBTN disable with timer [optional].

Only chance for firmware Update flow breakup is: Dropping the system or PBTN Override disable.

A mechanism to continue the firmware update process is needed if it gets interrupted in the middle and this scheme is called **seamless recovery** as it would be done without user interventions. This feature heavily relies on the Fault tolerant update scheme that's outlined in the previous sections. During FW update, TXE would have 1 copy of CC-OLD or CC-NEW to ensure hand off to IBB. IBB which tracks and handles the FWUPDATE/Recovery must track the interrupted FW update and continue the update process.

Figure 24-17. Seamless Recovery Flow



24.10 Firmware Boot Partition Protection

Firmware boot partition must be Write protected before IAFW starts running the any 3rd party driver/apps i.e. it exits PMAUTH Stage. Both EMMC and UFS enable boot partition protection feature. IAFW must disable PERMANENT_WP feature on both UFS and EMMC as this could be misused by a malware to cause a DOS on the firmware update onto boot partition.

24.10.1 EMMC Boot Partition PO Protection

In order to allow the host to protect the boot area against erase or write, the eMMC shall support two levels of write protection for each of the boot areas: Permanent write protection or power-on write protection. This protection can be applied to each boot area individually or to both regions at the same time using the BOOT_WP register (EXT_CSD [173]).

Production platform must not expose the RESET PIN to host and it is even advised not to have a RESET PIN for EMMC device RESET.

APL Platform firmware shall only enable Power-on WP on the Boot partition.



24.10.2 UFS Partition PO Protection Feature

Permanent write protection (permanent: once enabled it cannot be reversed): IAFW must disable this feature on the LUN that corresponds to the boot partitions.

Power on write protection (write protection can be cleared with a power cycle or a hardware reset: IAFW must enable this before running 3rd party apps/drivers from within IAFW. Platform must ensure not to allocate RESET pin for UFS device reset.

24.10.3 FW Update Types and Use Cases

24.10.3.1 Full IFWI Update

The Entire FW image on the NVM is updated. There are several sub-cases:

1. In manufacturing. The FW update can occur during initial manufacturing if a different FW must be applied from the image initially programmed. This may also occur when in refurbish / repair scenario.
2. In the field (production) via OEM. The FW update occurs in the field. A safe (power fail tolerance) update process is critical to the end user experience of reliability of the platform.

24.10.4 Capabilities

The following capabilities are used to achieve the FW update use cases:

1. **SPI Flash Protection Override Strap (SPI only):** When this HW strap is asserted the CSE Bringup enters a temporary disabled state. The HW allows the host to write to any area of the flash.
2. **IFWI Prepare for update** - Prepares CSE for update by placing platform in recovery mode and disables NVM access and invalidates ICV blobs.
3. **Data Clear:** Allows for reset/clearing of the CSE data region. CSE data is separate from CSE code for Layout 2.0 (RPMB for eMMC/UFS, Device expansion region for SPI). This is needed for refurbish and repair scenarios. There is a IFP value for policy of allowing Data clear. There is an option to only allow Data clear if a secure token is used to authorize it. This command can only be entered if CSE FW is in recovery. Recovery is entered by either IFWI prepare for update or if CSE Non Critical partition BPDT is invalid, or if CSE errors out during initialization.
4. **Security Version Number:** The security version number is used to ensure the FW will not be downgraded to a security version that is known to have vulnerabilities.
5. **Separate partitions:** The FW is separated into distinct partitions where a partition defines a group of FW that is updated together. All FW modules in that partition come from the same FW build. This defines a unit of compatibility where all FW in that partition was tested together. Each partition has a single manifest. If the manifest is valid the FW loader assumes all modules in the partition are valid and if not the loader will enter recovery mode.
6. **Manifest partition hash:** The manifest partition hash provides a complete hash of the downloaded partition. This enables quick verification by the FW update FW before the partition is written to flash.



7. **SPI controller flash region protection ranges:** The SPI controller provides a capability for the FW to specify ranges of the CSxE flash region that can be locked down such that the HW will deny write and erase operations to these flash regions. The lock is released at the next CSxE reset (not PG reset).

24.11 FW Update Requirements

Table 24-1 FW Update Requirements

Requirement Title	Requirement Description	Requirement Rationale
FW upgrade	The FW shall support update to newer versions of FW compatible with the platform.	Product life cycle sustainability.
FW downgrade	The FW shall support downgrade to older versions of FW compatible with the current version.	OxM requirement for product life cycle sustainability.
Disallow FW downgrade to lower Security Version Number	The FW shall not permit FW downgrade to FW whose security version number is less than the current security version number.	Security requirement to prevent host sw from re-introducing security vulnerabilities via the FW update interface.
FW Update power loss survival	The FW must be able to boot the system to normal OS environment if a power loss occurs at any time during a FW update.	FW update robustness.
Data Clear	Allows for clearing of CSE Data region for Layout 1.5/RPMB for layout 2.0	OxM requirement for CSxE FW reflash/refurbish on the manufacturing floor

24.12 FW Architecture

24.12.1 Code Partition Types

There are 3 main regions of the CSE region for IFWI 1.5. Refer IFWI Layout for more details, but here are some details to clarify the flow.

- CC- Critical Components which is essential to boot IAFW which enables recovery; the list includes CSE RBE/CSE BUP/PMC patches/SMIP.

Note: This maps to Fault Tolerant Partition in the past.

- NCC 1 and 2-Non Critical Components which are essential for system boot but not needed for recovery. These regions includes Intel FW components like CSE Main FW, ISH FW and so on.
 - This includes IUPs and NFTP (Non Fault Tolerant Partitions from SPT baseline)



24.12.2 Boot Partition Descriptor Table

The Boot Partition Descriptor Table (BPDT) is a table of offsets to all individual sub-partitions contained within the Logical Boot partition LBP. A sub-partition is defined as a signed data block with the manifest available at the beginning allowing authentication.

Refer Apollo Lake IFWI layout document for further details.

24.12.3 FW Versioning

FW update allows update to newer versions of the FW and downgrade to older versions of the FW except as excluded by specific criteria. The following versioning information is contained in partition manifests and is used:

- **FW Version:** The FW version number of format typically written in format of major.minor.hotfix.build with the following usage.
 - **Major:** The major version of the FW. This number is incremented for new chipsets of the same product class. For example Broadwell PCH class products are major number 10 and Skylake PCH class products are major number 11, and SoC class products use different major number as they started from a different lineage.
 - **Minor:** The minor version of the FW. This number is incremented when significant new capabilities are introduced into an existing major version of FW. This typically creates a branch in the FW source project.
 - **Hotfix:** The hotfix version of the FW. This number is incremented when fixes are applied to a branch of the FW.
 - **Build:** The build number of the FW. This number is inserted by the FW build system for each FW build. Customers may observe gaps in build numbers between customer releases because internal builds also increment the build number.
- **Security Version Number:** The security version number (SVN) is incremented when problem is fixed in the FW that requires a TCB recovery operation. The TCB recovery operation is a significant event in the life cycle of the FW requiring renewing of many of the keys used by the FW; incrementing of the SVN is qualified by an Intel PRT. The SVN is scoped to the product's FW signing keys.
- **Version Control Number:** The version control number (VCN) is incremented whenever a change is made to the FW that makes it incompatible from an update perspective with other previously released versions of the FW. Note to readers familiar with past ME concepts of blacklists, version history, and data format versions, etc – an RCR replaced these mechanisms with the simpler VCN mechanism. The VCN is scoped to the product's FW signing keys
- **Production Flag:** The production flag indicates this FW is a production release. This flag is set when the production build criteria are met for the FW. A FW update from a FW with the production flag set to a FW with the production flag clear is not allowed. Note that simply qualifying as being signed by the production key is not sufficient as there are pre-production versions of the FW that are signed with the production key for various validation and sampling reasons.
- **Previous SVN:** Data structure outside files system used to track previous CSE SVN. This is needed for data migration. ROM uses this value to determine if it should generate previous root key based on SVN for data migration after FW



update has been completed. DnX also looks at this value to see if SVN downgrade has occurred.

- For ME 8.0 it was set and cleared by FW update process
- For SPT FW update set it, but there was a bug where it was not cleared.
- For BXT and beyond VFS is responsible for setting and maintaining PSVN. It is no longer cleared. It is set after the CSVN blob is read.
- Current SVN - SVN of RBE that has currently loaded.

Examples of various scenarios and proposed versioning actions are described for illustrative purposes in the following table.

Table 24-2. Version Scenarios and Actions

Scenario	Versioning Action(s)
First final production release of FW	Set production flag
Bug fix that does not break compatibility with previous FW versions.	Increment hotfix
Bug fix that cannot be downgraded to once fixed (a.k.a historically known as critical hotfix)	Increment VCN Increment hotfix
New feature / capability introduction (historical example: point release of DAL in Cougarpoint)	Increment minor
Bug fix that requires a TCB recovery operation.	Increment SVN Increment VCN Increment hotfix
FW change that breaks compatibility with previous FW releases	Increment VCN Increment hotfix

Validation recommendations: FW versions that are compatible with one another are determined by the update image having a VCN that is greater than or equal to the currently installed FW version. Therefore to qualify a new FW build, validation should run necessary tests to reach desired coverage against only previous FW builds whose VCN is equal to or greater than the new FW build's VCN.

24.12.4 Upgrades and Downgrades

The FW supports upgrades (update to a new version of FW) and downgrades (updates to a lower version of FW) according to the following flow:

- 1) If update image's major != current major, it is an incompatible FW image.
- 2) If update image's minor.hotfix.build == current minor.hotfix.build, it is same FW.
- 3) It is a FW upgrade
 - a. If minor > current minor, or
 - b. If minor == current minor and hotfix > current hotfix, or
- 4) Else it is a FW downgrade.

Note: There is no enforcement of these values in Host/IAFW based updates.



24.12.5 Data Format Sustaining Considerations

Refer IFWI layout documentation. CSE Code and data have been separated into two different SPI regions or between Boot partitions and RPMB for eMMC/UFS.

It is **strongly recommended** that post production FW releases do not change data format of variables in such a way as to cause an incompatibility that breaks FW upgrade or downgrade. The following recommendations are made with respect to data format changes, if they are necessary:

- Upgrade FW (higher version number) should place new fields in either:
 - New data variables (preferred)
 - Reserved fields of existing data variables if it can be confirmed that all previous FW ignored those reserved fields.
- Upgrade FW should always keep FW variables that are used by older FW in a consistent and coherent state such that a FW downgrade will result in the downgraded FW operating within correct known parameter settings upon downgrade.
- If the above two guidelines cannot be met the VCN should be incremented to prevent downgrade to FW that is incompatible with previous FW.

24.12.6 SVN/VCN upgrade/downgrades

CSE's SVN and VCN are mechanisms that help prevent a downgrade to security or functionality compromised CSE FW.

24.12.7 Update_IMAGE_CHECK

This is an API that CSE will provide to help host determine SVN checks on IFWI image components. There is a VCN check in addition for SVN CSE components.

What this provides:

1. Mechanism to notify if IFWI components' SVNs are being downgraded
2. Mechanism to notify BIOS that a rollback is possible after update (roll back should fail if CSE SVN or VCN has been incremented due to incompatible/unreadable data).
3. Mechanism to notify BIOS that of CSE VCN is being downgraded and may not correctly work.

What this does NOT provide

1. This is not a comprehensive check that will be done when CSE loads modules. This is a quick check on SVN/VCN and to note if a rollback would be possible on CSE side.
 - a. Any checks specific to component will note be included (example PMC, ISH)
 - b. Build version differ on CSE components would not be checked

24.12.7.1 Update IMAGE CHECK Details

This checks the SVN of the incoming image in memory against current image on flash



Host will pass in a pointer to where the update image is in memory.

IMPLEMENTATION NOTE: CSE must use RSO for access to the memory addresses provided. Otherwise, CSE might accidentally access its own device space, and this can be used for attacks.

BUP loader will perform verification checks and provide the appropriate error to host.

BUP Loader must verify SVN's of all non-CSE IFWI components in the update image compared against what is in the ARB SVN file. If anti-rollback is disabled, then skip this check for non-CSE IFWI components.

For all CSE components check against the VCN and SVN of the SPIE (Signed Package Info Extension) Header of the RBE. CSE VCN of the RBE of the update image checked against what is loaded in the current RBE manifest. VCN check should be against both copies of RBE (BPDT1 and 2 if it exists in both places)

If new image CSE's SVN / VCN > current one, BUP must set "rollback not possible" flag in the response.

Note: This is not available after POST due to concerns for having DMA from host to CSE after EOP as well as having a single unified flow.

24.12.8 Data Clear

In the IFWI Prepare for Update flow, Data Clear can be used optionally. When Data Clear is used, it clears all TXE & BIOS data stored in device expansion region on SPI boot devices and RPMB region on EMMC boot devices. This can be used during manufacturing / refurbishing scenarios.

CSE accepts 'DATA CLEAR' command only after successful execution of 'IFWI PREPARE FOR UPDATE' Command.

If Data Clear is not applied then Data Clear Lock can be send before IFWI Prepare for update as per flow.

For more details of sending Data Clear command, refer its MKHI command details and 'Data Clear on TXE Enabled System' section under Security Engine chapter.

25 TCO

The only TCO feature supported by APL SoC is the WDT.

25.1 TCO IO Space

The TCO I/O registers reside in a 32-byte range that starts 96 bytes above the power management (ACPI) I/O space. Thus TCOBASE = ACPIBASE + 0x60.

25.2 Watchdog Operation

If the TCO watchdog timer is not reloaded within 2.4 seconds after boot, it will generate a TCO SMI. This will generate an SMI if the TCO_EN, ACPIBASE + 0x40[13], and GBL_SMI_EN, ACPIBASE + 0x40[0], bits are set.

If the TCO watchdog timer is not reloaded for another 2.4 seconds and the No Reboot bit (PBASE + 0x08[4]) is not set, then APL SoC will reboot.

After booting, the IA FW must disable the TCO logic within 4.8 seconds or this reset will reoccur. The IA FW should disable the TCO watchdog timer from resetting the system by setting the NO_Reboot bit. The OS or device driver will be responsible for resetting this bit after the TCO driver loads.

The IA FW must always service the SMI by clearing the TCO_STS (writing '1b' to ACPIBASE + 0x44[13] and the TIMEOUT bit (writing '1b' to TCOBASE + 0x04[3]), whenever TCO SMIs are enabled. The IA FW must not reset the watchdog timer in the SMI handler; this must be left to the OS present TCO driver. The IA FW may log the TCO timeout, if desired.

If The TCO_TMR_HALT bit and TCOBASE + 0x08[11] are set, the TCO timer will be halt. Since TCO SMIs can be undesirable in some implementations, especially if the timer reload software had not been enabled, this provides a way for the software to halt the TCO timer.

The OS may require the IA FW to expose details of the hardware Watchdog Timer. For this purpose, Microsoft® has specified a Watchdog Timer Resource Table (WDTRT) in their "Requirements for Hardware Watchdog Timers Supported by Microsoft® Windows®" design specification. The WDTRT is a fixed resource table, defined according to the ACPI Specification. Refer the Microsoft* design specification for details on the format of the WDTRT.

25.2.1 Second Timeout STS

The SECOND_TO_STS bit (TCOBASE + 0x06[1]) is set to indicate that the TIMEOUT bit has been set or is currently set and a second timeout occurred before the TCO_RLD register, TCOBASE + 0x00 was written. During POST the IA FW should make sure that the SECOND_TO_STS bit is cleared (writing '1b' to TCOBASE + 0x06[1]) before transferring control to the OS.



25.3 Checking TCO Timer after Reset

The state of the TCO timer and its status bits (SECOND_TO_STS and TIMEOUT) must be checked after a reset to enforce proper operation of the TCO features.

If the SECOND_TO_STS bit and the TIMEOUT bit are both set, then the IA FW knows a system reboot has occurred and the third TCO timeout has already occurred.

If the SECOND_TO_STS bit is set and the TIMEOUT bit is clear, then the IA FW knows the system reboot occurred due to an OS lockup and the third TCO timeout has NOT occurred yet. The IA FW should clear the SECOND_TO_STS bit and then work to prevent TCO timeouts.

The IA FW is counting down since the default is TCO timer on. The IA FW should perform one of the following measures to prevent a TCO timeout:

1. Halt the TCO timer by setting the TCO_TMR_HLT bit, TCOBASE + 0x08[11].
2. Set the TCO_TMR, TCOBASE + 0x12, to a larger max count (some value larger than the default value, which is 0x04 or ~2.4 seconds) and then force a TCO timer reload (by writing to the TCO_RLD register, TCOBASE + 0x00) regularly.

25.4 Additional Settings

The TCO_LOCK bit (ACPIBASE + 0x68 [12]) should be set in the TCO1_CNT register in order to prevent writes from changing the TCO_EN bit (in offset 0x40 of ACPI I/O space).

§

26 Legacy Free Systems

26.1 Legacy Free Systems Handling by IA FW

"Legacy Free" as referred to in this section implies the removal of Super I/O (PS/2 Keyboard, PS/2 Mouse, COM Ports, Parallel Ports, Super I/O based floppy), and ISA bus removal. Typically these devices are not truly Plug and Play (PnP) compliant because the IA FW has to explicitly report to the OS about the usage of resources (I/O, Interrupt, DMA and so on.) for these devices. This section discusses some of the IA FW implications when dealing with "Legacy Free Systems"

26.2 Generic Address Structure

The Generic Address Structure describes the location and access methods of a hardware register. Refer ACPI specification for more information.

Table 25-1. Generic Register Address Structure

Field	Byte Length	Byte Offset	Description
Address_Space_ID	1	0	<p>The address space where the data structure or register exists. Defined values are:</p> <ul style="list-style-type: none"> 0 System Memory 1 System I/O 2 PCI Configuration Space 3 Embedded Controller 4 SMBus 5 to 0x7E – Reserved 0x7F Functional Fixed Hardware 0x80 to 0xBF – Reserved 0xC0 to 0xFF – OEM Defined
Register_Bit_Width	1	1	The size in bits of the given register.
Register_Bit_Offset	1	2	The bit offset of the given register at the given address.
Access_Size	1	3	<p>Specifies access size.</p> <ul style="list-style-type: none"> 0 Undefined (legacy reasons) 1 Byte access 2 Word access 3 Dword access 4 Qword access

Field	Byte Length	Byte Offset	Description
Address	8	4	The 64-bit address of the data structure or register in the given address space (relative to the Processor). Refer Table 27-2 for specific formats.

Table 25-2. Address Space Format

Address Space	Format											
0 – System Memory	The 64-bit physical memory address (relative to the processor) of the register. 32-bit platforms must have the high DWORD set to 0.											
1 – System I/O	The 64-bit I/O address (relative to the processor) of the register. 32-bit platforms must have the high DWORD set to 0.											
2 – PCI Configuration Space	PCI Configuration space addresses must be confined to devices on PCI bus 0. The format of addresses is as follows: <table border="1"> <thead> <tr> <th>WORD Location</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Highest WORD</td> <td>Reserved. Must be 0.</td> </tr> <tr> <td></td> <td>PCI Device number on bus 0.</td> </tr> <tr> <td></td> <td>PCI function number.</td> </tr> <tr> <td>Lowest WORD</td> <td>Offset in the configuration space header.</td> </tr> </tbody> </table> For example: Offset 23h of Function 2 on device 7 on bus 0 would be represented as: 0x0000000700020023.		WORD Location	Description	Highest WORD	Reserved. Must be 0.		PCI Device number on bus 0.		PCI function number.	Lowest WORD	Offset in the configuration space header.
WORD Location	Description											
Highest WORD	Reserved. Must be 0.											
	PCI Device number on bus 0.											
	PCI function number.											
Lowest WORD	Offset in the configuration space header.											
0x7F–Functional Fixed Hardware	Use of GAS fields other than Address_Space_ID is specified by the CPU manufacturer. The use of functional fixed hardware carries with it a reliance on OS specific software that must be considered. OEMs should consult OS vendors to ensure that specific functional fixed hardware interfaces are supported by specific operating systems.											

26.3 Debug Port Table

This optional table specifies an independent Debug Port for debugging purposes. If not present, the operating system will revert to existing debugging devices. Systems indicate the address of the debugger port in the Debug Port Table. This table is located in system memory with other ACPI tables. It must be referenced in the ACPI RSDT table.

The presence of the Debug Port Table indicates that the system includes a Debug Port. The contents contain information about the configuration of the Debug Port. Refer the Debug Port Specification for more details.

Table 25-3. Debug Port Table Format

Field	Byte Length	Byte Offset	Description
Header			
Signature	4	0	'DBGP'. Signature for the Debug Port Table.
Length	4	4	Length, in bytes, of the entire Debug Port Table.
Revision	1	8	1
Checksum	1	9	Entire table must sum to zero.
OEMID	6	10	OEM ID.
OEM Table ID	8	16	For the Debug Port Description Table, the table ID is the manufacturer model ID.
OEM Revision	4	24	OEM revision of Debug Port Description Table for supplied OEM Table ID.
Creator ID	4	28	Vendor ID of utility that created the table. For the DSDT, RSDT, SSDT, and PSDT tables, this is the ID for the ASL Compiler.
Creator Revision	4	32	Revision of utility that created the table. For the DSDT, RSDT, SSDT, and PSDT tables, this is the revision for the ASL Compiler.
Interface Type	1	36	Indicates the type of the register interface: 0 = full 16550 interface. 1 = 16550 subset interface compatible with Microsoft* Debug Port Specification. 2-255 = reserved.
Reserved	3	37	Must be 0.
BASE_ADDRESS	12	40	The base address of the Debug Port register set described using the Generic Register Address Structure.

26.4 Legacy-Free Handling in ACPI Summary

The IA FW has to implement the Legacy-Free support as defined below:

LEGACY_DEVICES flag is set to "0" in the ACPI FADT table

RESET MECHANISM is defined as indicated in the above section(s)

8042 flag of Boot Architecture Flags in ACPI FADT table is set to "0" for systems that do not implement the Keyboard/Mouse controller (Super I/O). On some mobile



systems the 8042 flag can be set to a "1" if the Keyboard/Mouse controller is still implemented.

Debug Port Table is defined in the IA FW as described above.

Plug and Play Detection should not report any usage of Super I/O resources

Keyboard/Mouse Controller – I/O ports 60 and 64

COM – I/O Ports 2E8-2EF, 2F8-2FF, 3E8-3EF, 3F8-3FF

LPT – I/O Ports 278-27F, 378-37F, 3BC-3BF

SoundBlaster® – I/O Ports 220-22F

Joystick/Game – I/O ports 200-20F

MIDI – I/O ports 300-301

Floppy – I/O Ports 3F0-3F7

The IA FW should use the use the USB legacy Keyboard and Mouse Software IRQ generation support as described above in the USB chapter.

IA FW ROM Update capability from the USB Floppy instead of a regular Super I/O floppy interface should be provided.

Software interrupts defined below and their sub-functions should still be supported

INT8 – System Timer

INT9 – Keyboard Data

INT10 – Video BIOS interface

INT11 – Equipment detection

INT13 – Hard disk interface (Large hard drive support)

INT15 – For keyboard and mouse support and other memory reporting capability

INT16 – Keyboard control

INT19 – Bootstrap loader

INT1A – RTC and P PCI BIOS Routines

INT1B – CTRL+BREAK handler

INT23 – CTRL+BREAK and CTRL-C handler

§



27 Debug Configurations

This chapter describes the configurations needed for different debug features on Intel Processors.

27.1 Debug MSR

APL Processors support the Debug MSR for controlling the enable/disable of debug features (via JTAG/TAP port) on the processor. Specifically the enabling of this interface controls the following

1. Ability to load debug patches
2. Ability to run the processor in probe mode
3. VCU API functions

27.1.1 IA FW Configurations of DEBUG MSR

IA FW should verify the presence of this MSR by confirming that CPUID.(EAX=1):ECX[11] is set before accessing this MSR. If set

1. System IA FW must set bit 0 of this MSR to enable Debug Interface. This bit must be set by IA FW before the first SMI occurs.
2. The content of this MSR may be locked by System IA FW by setting bit 30. If IA FW fails to set the lock bit, it is automatically set by the processor hardware on assertion of the first SMI.
3. Bit 31 of this MSR indicates the status if any debug may have occurred e.g. Debug enabled and then disabled prior to the first SMI or the register being locked. The value is read again by the processor on every reset and is cleared only by removal of coin battery

27.2 Intel Processor Trace

27.2.1 Intel Processor Trace Storage in Memory

Storage location of instructions is handled via Single Range output scheme or Table of Physical Addresses (ToPA) scheme. The availability of Intel Processor trace feature is known by reading CPUID(EAX=7):EBX[25]. This CPUID enumeration when set, indicates the availability of Intel PT feature.

27.2.1.1 Single Range Output Scheme

In single range output, all the trace output is stored in a contiguous memory location. The memory range behaves like a circular buffer storage area for trace i.e. after the last entry, the trace output wraps around to the beginning of the memory location.

The single range output scheme is used by default when ToPA scheme is not enabled. It is recommended to use this scheme in scenarios where a single large chunk of trace output is required to debug a system hang or when the trace output needs to be stored in the debug port MMIO region.

27.2.1.2 IA FW Configurations for Intel Processor Trace Single Range Output Scheme

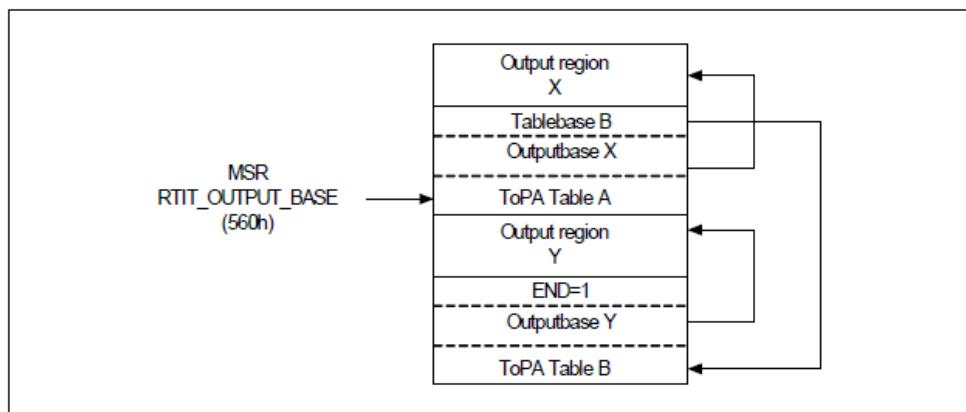
1. System IA FW must read CPUID(EAX=20):ECX[2] and proceed to enable Single Range output scheme only if it is set.
2. To enable Intel Processor Trace Single Range output scheme, System IA FW must clear bit 8 in MSR IA32_RTIT_CTL (570h).
3. System IA FW must configure the base address for the contiguous memory location in MSR IA32_RTIT_OUTPUT_BASE (560h) [47:7].
4. The corresponding mask bits for the base address need to be configured by IA FW in MSR IA32_RTIT_OUTPUT_MASK_PTRS (561h) [31:7]. If any base address bits are set within the masked range, an error will be signaled, and tracing will be disabled.

Note: System IA FW can allocate a single output trace memory range between 128 bytes to 4GB in size.

27.2.2 Table of Physical Address (ToPA) Scheme

Unlike the Single Range output scheme, in ToPA scheme a linked list of pointers to memory regions is used to store the output trace. This linked list structure is created by IA FW and each structure can point to a 4 kB aligned region address.

The below figure is an example of linked ToPA structures in memory. MSR IA32_RTIT_OUTPUT_BASE (560h) points to the ToPA table A which in turn has a pointer to the first output memory region X. The last entry in ToPA table A points to the next table Table B. Table B has a pointer to the second output memory region Y. Since Table B is the last ToPA table in the list, it will point back to the first entry which is Table A.



27.2.2.1 IA FW Configurations for Intel Processor Trace ToPA Scheme

IA FW can enable ToPA scheme only if CPUID(EAX=20):ECX[0] is set.

Based on the ToPA structure parameters as shown below, System IA FW needs to configure different values. When any of the below parameters are set incorrectly by IA FW or when unsupported entry is entered for size or if Stop or Int are not cleared when End is set, then error bit is set in MSR IA32_RTIT_STATUS(571h).

Table 26-1. Intel Processor Trace Structure in Memory (Sheet 1 of 2)

Bit Field	Description
63:48	Reserved
47:12	Output Region Base Physical base address: If END=0, this is the 4K-aligned base physical address of the DRAM output region specified by this entry. If END=1, this is the base physical address aligned to the size in bits [9:6] of the next ToPA table (which could be the base of the current table, a circular buffer).
11:9	Reserved
9:6	Size: Indicates the size of the output region in bytes with following encoding. 0000 - 4k, 0001 - 8k, 0010 - 16k, 0011- 32k, 0100 - 64k, 0101 - 128k, 0110 - 256k, 0111 - 512k, 1000 - 1 M, 1001 - 2 M, 1010 - 4 M, 1011 - 8M, 1100 - 16M, 1101 - 32 M, 1110- 64 M, 1111 - 128 M The output region base address defined in bits 47:12 must be aligned to the size defined in this field. The value defined in this field is valid only if END=0.
5	Reserved
4	Stop: When the output region indicated by this entry is filled, trace is disabled.
3	Reserved
2	Int: When the output region indicated by this entry is filled, Perfmon LVT interrupt is triggered.
1	Reserved
0	End: When this value is set, it indicates the last ToPA entry and the output region base address holds the table base address.

For each processor thread, IA FW must create the ToPA table of n entries as per below requirements. Since APL processor supports multiple ToPA entries, System IA FW can select the number of entries depending on user requirement or total memory needed. The recommended default trace size for APL processors is 2 MB. Each structure in the table is 16 bytes size.

1. The first n-1 entries in the table should have the trace output base address in bits 47:12, bits 4,2 and 0 must be cleared.
2. The last entry in the table should have the base address of the table location in bits 47:12, bits 4 and 2 must be cleared and bit 0 must be set.

Note: It is advisable to use larger output regions, in order to reduce the performance impact of transitioning from one to the next. When ToPA scheme is enabled, to avoid reuse of values from previously enabled Single Range output scheme, IA FW must clear the value of MSR IA32_RTIT_OUTPUT_MASK_PTRs (561h)

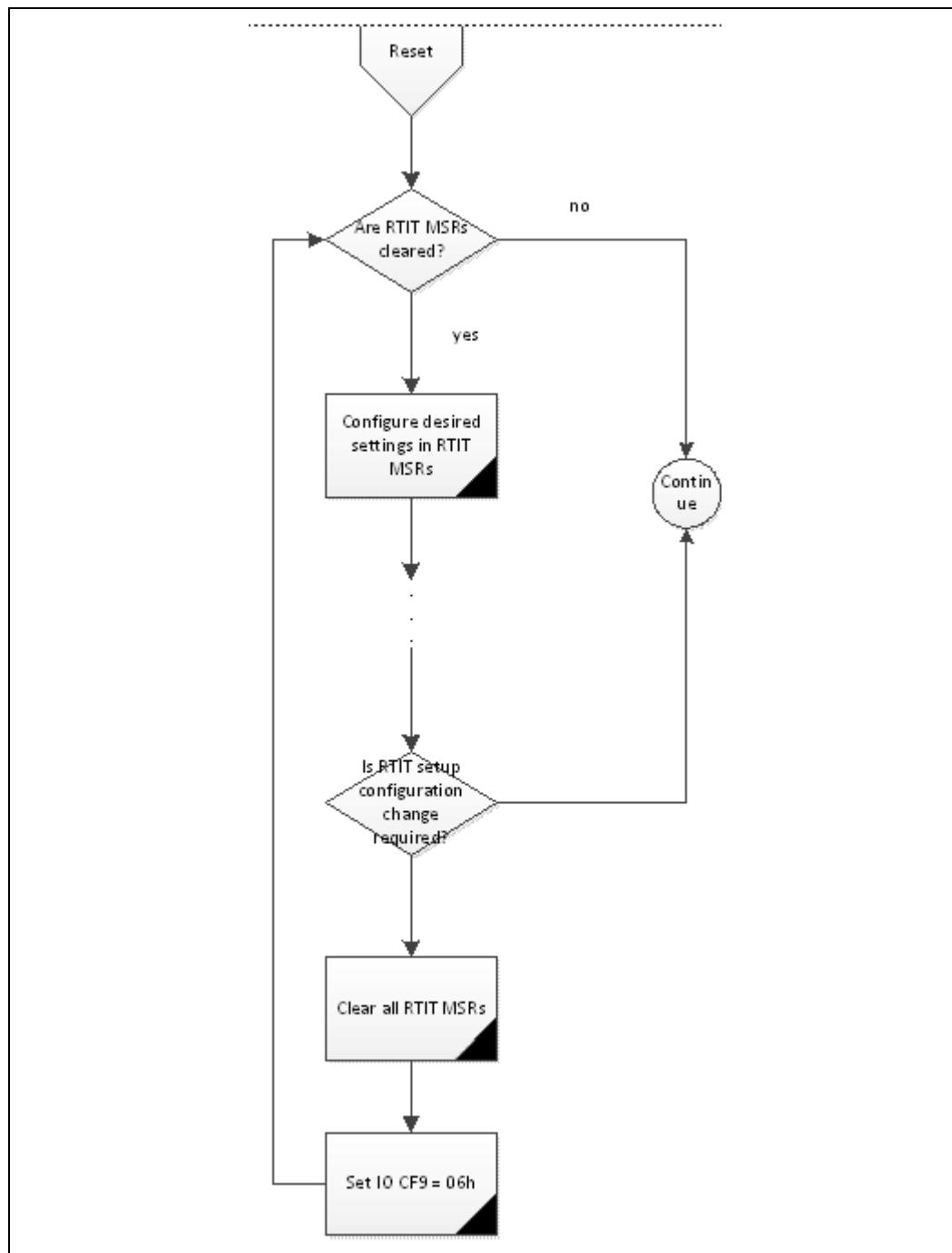


27.2.3 Other System IA FW Requirements for Intel Processor Trace

For every processor thread,

1. Prior to initialization of Intel Processor Trace
2. IA FW must clear MSR IA32_RTIT_CTL (570h) bit 0.
3. IA FW must clear MSR IA32_RTIT_STATUS (571h) to all zeros.
4. IA FW needs to allocate and create the necessary memory and structures for Intel Processor Trace. A memory size of at least 4 kB per logical thread is required for storing trace data.
5. For both Single output range as well as ToPA scheme, System IA FW must configure the trace address base in MSR IA32_RTIT_OUTPUT_BASE (560h) bits ((MPA-1):7). The Maximum Physical Address for the processor can be determined by CPUID.(EAX=0x80000008):ECX[7:0]. System IA FW must configure the output base address accordingly i.e. IA32_RTIT_OUTPUT_BASE bits ((MPA-1):12).
6. IA FW must be able to support all the following modes for Intel Processor trace enable
7. Configuration and setup of
8. Single Range Output Scheme as described in [Section 24.2.1](#) and steps above And/Or
9. Table of Physical Address scheme as described in steps above.
10. Configuration of either/both the schemes as described above and enables trace by setting.
11. Bits 0,2,3,8 and 13 to 1b to enable RTIT ToPA method and
12. Bits 0,2,3 and 13 to 1b to enable RTIT Single range output scheme method
13. (Default) Not do any of the above.
14. RTIT MSR values are retained over a warm reset to preserve the previous trace state in case of system hang or unexpected system reset. It is hence recommended that during boot flow, System IA FW configure the RTIT MSRs only if there is no previously configured value. When IA FW gets a request to change the RTIT configuration in setup, it can clear all the RTIT MSRs before issuing a system reset.
15. IA FW may enable Trace Filter by setting MSR IA32_RTIT_CTL (570h) [39:36] = 010b. The required trace filter range must be configured by IA FW in MSRs IA32_RTIT_ADDR0_A (580h), IA32_RTIT_ADDR1_A (582h), IA32_RTIT_ADDR0_B (581h) and IA32_RTIT_ADDR1_B (583h).

Note: Even though the RTIT MSRs are persistent over warm reset, it is recommended that System IA FW reconfigure all the MSRs unless requested not to.



27.2.4 Intel Processor Trace and SMM

On every #SMI IA32_RTIT_CTL.TraceEn will be saved in SMRAM and then cleared there by disabling the trace feature. When RSM is executed to return from SMM mode, the IA32_RTIT_CTL.TraceEn value will be restored from SMRAM. This allows tracing to resume upon re-entry to non-SMM mode.

If IA FW wishes to enable tracing while in SMM, it must execute the following steps.

1. On entering SMM enable Intel Processor Trace by setting MSR IA32_RTIT_CTL (0x570) TraceEn bit[0]

Before exiting SMM i.e., RSM instruction disable Intel Processor Trace by clearing MSR IA32_RTIT_CTL (0x570) TraceEn bit[0]

Note: This will utilize the same trace configuration and output as the non-SMM trace.

If IA FW wishes to trace SMM code separately, it should do the following:

On entry to SMM

1. Save all Intel Processor Trace MSR state to memory.

Load new Intel Processor Trace MSR configuration (perhaps from a saved MSR image in memory).

Set IA32_RTIT_CTL (0x570) TraceEn bit[0].

Before exit from SMM

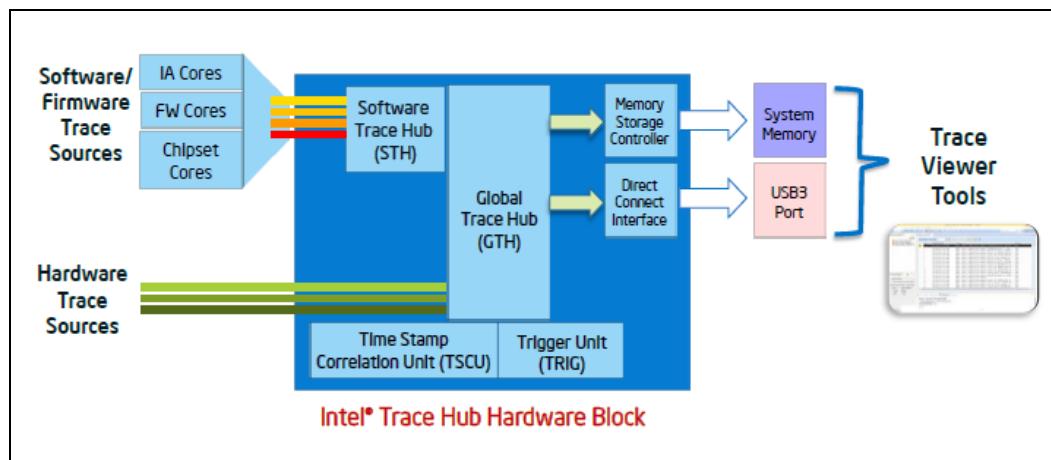
1. Clear IA32_RTIT_CTL (0x570) TraceEn bit[0].
2. Save SMM Intel Processor Trace MSR state to memory, taking care not to overwrite the non-SMM MSR state.
3. Restore non-SMM Intel Processor Trace MSR state from memory

This will ensure that the non-SMM trace is not corrupted by the SMM trace.

Note: On all APL processors with CPUID(EAX=1):EAX equal to 000406E0h, System IA FW must clear MCHBAR register 5138h[13] to 0b.

27.3 Intel® Trace Hub (Intel® TH)

Intel® Trace Hub (codename: NorthPeak) is a hardware block, embedded inside Silicon. It can collect debug trace data from different sources in the system and combine them into a single output stream with time-correlated to each other. The sources including hardware, software, firmware and IA FW/BIOS, it is capable for Platform level debug. It uses USB3 port as primary trace data outputs path for closed chassis debug.





NorthPeak is PCI device (B0/D0/F2) in APL which has an extra fourth 64-bit 2MB BAR called FWTMR. This fourth BAR must be programmed in NPK and PSF1 via IOSF-SB by IA FW and its PSF configuration object hidden as an ACPI function.

27.3.1 Intel® NorthPeak Initialization Flow

1. North Peak device is power gated using the following two PMC bits (NPKVRCEN and NPKVNNEN), if debug is needed early on, then the PMC will set these bits to enable the NorthPeak power domains.
2. PCI Enumeration for NorthPeak device (Bus 0, Function 0, Device 2)
3. Discover the NPK Device with Vendor ID 0x8086 and Device ID 0x5A8E.
4. Program the MTB Base Address Register.
5. Program the SW BAR.
6. Program the RTIT BAR.
7. Program the FW BAR.
8. Set the memory space (CMD bit 1) to claim memory access.
9. A shadow PCI device within the backbone PSF needs to be programmed with the value of FW BAR, have its memory space enabled, and then hide the shadow device by below steps:
 10. Program Lower Base Address of FW BAR to PCR[PSF1] + 0x200
 11. Program Upper Base Address of FW BAR to PCR[PSF1] + 0x204
 12. Program PCR[PSF1] + 0x21C, Bit1 to enable Memory space.
 13. Program PCR[PSF1] + 0x238, Bit0 to hide the shadow device in OS.
 14. Next, IA FW should start enabling the various features that are given as IA FW options. The first part of this is to select the Switching Destinations of the HW/SW/FW trace sources, then IA FW will have separately initialization for Selected Output port.
 15. Once Selected output ports had been configured. Clear the Bypass bit (bit 0) in the DSC register and set Bus Master Enable (bit 2) in the CMD register.

27.3.1.1 Switching Destinations Register Set Up

Master No.	Master Name	Register	Address Offset	Bits
15	RTIT	SWDEST[1]	CSR_MTB_LBAR + 0xCh	[31:28]
16	CSE	SWDEST[2]	CSR_MTB_LBAR + 0x10h	[3:0]
17	CSE	SWDEST[2]	CSR_MTB_LBAR + 0x10h	[7:4]
18	CSE	SWDEST[2]	CSR_MTB_LBAR + 0x10h	[11:8]
19	ISH	SWDEST[2]	CSR_MTB_LBAR + 0x10h	[15:12]
22	HDMI Audio	SWDEST[2]	CSR_MTB_LBAR + 0x10h	[27:24]



Master No.	Master Name	Register	Address Offset	Bits
23	PMC	SWDEST[2]	CSR_MTB_LBAR + 0x10h	[31:28]
24	AET	SWDEST[3]	CSR_MTB_LBAR + 0x14h	[3:0]
27	Iunit	SWDEST[3]	CSR_MTB_LBAR + 0x14h	[15:12]
28	FTH	SWDEST[3]	CSR_MTB_LBAR + 0x14h	[19:16]
29	P-unit	SWDEST[3]	CSR_MTB_LBAR + 0x14h	[23:20]
256	IA FW	GWTDEST	CSR_MTB_LBAR + 0x88h	[3:0]

Program the SWDEST[n] value as 0x0, if switching destination is disabled.

27.3.1.2 Configure Memory as Output Port if Being Selected

1. Program the SWDEST[n] value as 0x8, if Memory is selected as output port.
2. For each MSC MSCnCtl register, program these bits MSCnLEN (Bit 10:8), MSCnMODE (Bit 5:4), WRAPENn (Bit 1), Set the MSCnEN (Bit 0)
3. Program the "CSR_MTB_BAR + 0C8h = 0x130000h" to enable the trigger overrides to for each of the trace sources that were selected.

27.3.2 Configure PTI as Output Port if Being Selected

1. Program the SWDEST[n] value as 0xA, if PTI is selected as output port.

For PTI_CTL register, set up the Pattern Generation Mode, Clock Divider, PTI mode and enable PTI by programming these bits: PATGENMOD (Bit 22:20), PTICLKDIV (Bit 17:16), PTIMODESEL (Bit 7:4), PTIEN (Bit 0)

Program the "CSR_MTB_BAR + 0C8h = 0x130000h" to enable the trigger overrides to for each of the trace sources that were selected.

27.3.3 Configure USB as Output Port if Being Selected

1. Program the SWDEST[n] value as 0x8, if USB is selected as output port.

27.3.4 USB3 Tracing Enable

Once NPK BAR has been programmed, the following registers need to be programmed for USB3 DbC Trace. All of those registers need to be written via IOSF-SB with XHCI ID 0xA2.

1. MTB0 offset is NPKBAR + 0x80000
2. Write MTB0 offset to DbC Trace IN Payload registers (offsets 0x50/0x54)
3. Write DbC Trace IN Payload and Status Qualifiers registers (offset 0x58, 0x68) with the static value of 0x00219000



4. DBCSTSCMD offset is NPKBAR + 0xA1008
5. Write DBCSTSCMD offset to DbC Trace IN Status registers (offset 0x60/0x64)

27.3.5 Post-MRC Initialization Flow

NPK

1. Does not autonomously power gate.
2. IA FW should still FuncDis the PCI config space in the fabric, and the hybrid ACPI function and the Native ACPI function.
3. IPC from IA FW to PMC to change state bits causing PMC to power gate NPK.
4. 25.3.5 USB 3.0 debug and Intel Trace Hub

27.4 DCI (Direct Connect Interface)

27.4.1 DCI Host Enable

For manufacturing mode system (before End Of Manufacturing is set), during cold boot and reset exit, BIOS can configure DCI(formerly known as EXI) based on user configured options.

To use USB 3.0 port for Kernel Debug through WinDbg, IA FW should disable DCI.

Intel recommends disabling the DCI interface in BIOS before shipping end products.

DCI Disable

To disable DCI, BIOS must perform the following to the ExI (aka DCI) IOSF-SB register (Offset 0x04h ExI Control Register ECTRL):

1. Set PCR[DCI] + 4h bit[4] = 0b

Actually, lock bit is at BIT0 and name of these 2 BITS are HOST_EXI_EN and HOST_EXI_EN_LOCK, ME_EXI_EN_LOCK and ME_EXI_EN are at DCI.EMEACC which is controlled by CSE and cannot be access by HOST.

2. Set PCR[DCI] + 4h bit[0] = 1b

Note that once bit [0] has been set to a '1', bit [4] may not be changed to a '1'. This sequence must be performed at each platform reset because the locking indication (bit [0] of the register) is not preserved across reset.

DCI Enable

To enable DCI via BIOS, BIOS performs the following to the DCI PCR register (Offset 0x04h DCI Control Register ECTRL):

1. Set PCR[DCI] + 4h bit[4] = 1b
2. Set PCR[DCI] + 4h bit[0] = 1b



Table 27-2. EMEACC Pertinent Fields

Bit	Access	Default Value	Description
4	RW/L	1b	ME EXI Enable (ME_EXI_EN): '0' – ME does not enable EXI. (Host may still enable EXI) '1' – EXI is enabled (unless overridden by EXI disable fuse) This bit cannot be written to when the ME EXI Enable Lock bit is set. This bit is resides in VNNAON and is only reset by pmc_exi_side_rst_b. NOTE: SAI protected
0	RW/L	0b	ME EXI Enable Lock (ME_EXI_EN_LOCK): When set, ME EXI Enable bit cannot be written to. When clear, ME EXI Enable bit can be written. This bit also locks itself when set. reset_type = pmc_cse_side_rst_b NOTE: SAI protected

Table 27-3. ECTRL Pertinent Fields

Bit	Access	Default Value	Description
4	RW/L	0b	Host-EXI enable (HEEN) when set to "1", allow BSSB-EXI (Boundary Scan Side Band, Extended flexible Interface) to be enabled when a BSSB-EXI adaptor is plug into USB3 port. When clear to "0", the host control is not enabling BSSB-EXI feature, but it is still possible for the ME to enable BSSB-EXI. This bit is resides in VNNAON and is only reset by pmc_exi_side_rst_b. Read or write to this bit is qualified by "EXI Lock" bit. When "EXI Lock" bit is "0", this bit is RW. When "EXI Lock" bit is set, this bit can only be cleared to zero by writing a 0 to it, write 1 has no effect.
0	RW/L	0b	Host EXI Enable Lock (HOST_EXI_EN_LOCK): When set, Host EXI Enable bit cannot be written to 1 but can be cleared. This bit is cleared by PLTRST#. This bit also locks itself when set. reset_type = PLTRST#.

**Table 27-4. EXIPCE Pertinent Fields**

Bit	Access	Default Value	Description
5	RW	0h	<p>Hardware Autonomous Enable (HAE) If set, then the IP may request a PG whenever it is idle and the EXI is disabled (ME/HOST EXI enable bit clear or fuse exi disable is set).</p> <p>Note: If this bit is set, then bits[2:0] must be '000'.</p>
0	RW	0h	<p>PMC Request Enable (PMCRE) if set, then IP will PG when idle and the EXI is disabled (ME/HOST EXI enable bit (IOSF SB private register) clear or fuse exi disable is set). and the PMC requests power gating by asserting the pmc_exi_sw_pg_req_b signal.</p>

27.4.2 EXI Power Gating Enable

To allow entry into S0i3, IA FW should include an option to clear EXI Host Enable, which will cause EXI to power gate and allow entry into the deepest power states.

IA FW also need to enable EXI autonomous power gating to allow S0i3 entry. These bits should be configured this way:

1. Set EXIPCE.HAE via IOSF EXI port 0xA8, offset 0x30, Bit5
2. Clear EXIPCE.PMCRE via IOSF EXI port 0xA8, offset 0x30, Bit0

27.5 GOP Debug

With debug version GOP, IA FW could get GOP logs by "UefiDebugLibDebugPortProtocol" protocol (typically via COM port), which would make it easier to get GOP log for further debug. IA FW should take care the debug port enable based on the HW design for the debug message output. Contact your Intel representative for debug GOP details.

27.6 Intel Crashlog

27.6.1 Overview

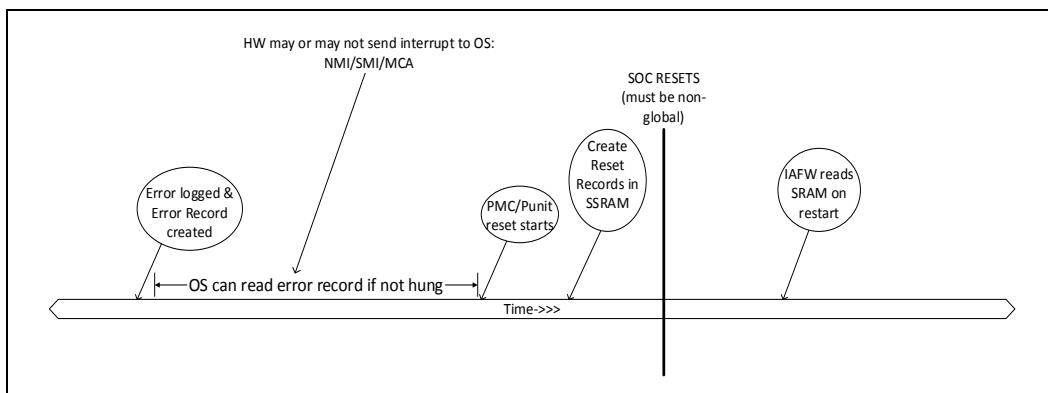
The Crashlog feature is intended for use by external customers (OEMs) as a means to triage and perform first level debug of failures. Additionally Crashlog enables BIOS or the OS to collect data on failures with the intent to aggregate the data and analyze failure trends. Crashlog effectively is a mechanism to aggregate debug information into a single location, and then allow access to that data via multiple methods, including target only IA FW/OS access.

Crashlog effectively has 2 components, the storage location, and the various agents that populate the storage location, which are typically firmware agents. For Apollo Lake and Apollo Lake, the storage location is the PMC Shared SRAM (SSRAM), which is

accessible via IOSF Sideband.PMC firmware, Punit firmware and CPU ucode populate the SSRAM area with Crash Data for each of these firmware agents.

The Crashlog region is structured into “records” and a “trace buffer”. Apollo Lake Crashlog has 3 types of records: “Global”, “Error”, and “Reset”.

Records are created at different times, with Global records being created once (often per reset), holding relatively static data. Error records are created on various error conditions, while Reset records are created on Reset entry. The SSRAM is preserved across warm and cold resets, so one of the main usage models is to have IA FW read the SSRAM after reset and process the Crashlog data for transfer to the OS. Below shows an example of the timeline of Crashlog record creation.



Intel Silicon/SoC has features which allow it to:

- Detect and notify the silicon/SoC subsystems of OS crash (pending platform reset) or other hang or critical error condition.
- Capture subsystem/firmware agent/microcontroller state before the pending reset.
- Use an SSRAM area, which retains its state across a reset.
- Allows the IAFW running on the AP (Application Processor) to:
 - Detect to crash, after the reset.
 - Access this SSRAM data.
 - Publish the SSRAM data in a defined format (ACPI/APEI BERT).
- By using the provided Windows* application to extract the ACPI/APEI BERT table into a file known as a “Raw Crashlog Data file”. This information can now be decoded and analyzed for potential Intel silicon/SoC detected failures.”

Refer Apollo Lake – Intel Crashlog Debugging Tool.

IA firmware running on AP to digest the hint provided by SOC on crash availability to grab the record and put it in a specific format which the Operating system component



understand. The Crashlog tool running on OS to grab the crash log information presented by the IA firmware by reading the ACPI BRET table

27.6.2 Crashlog SSRAM Organization and Size

The PMC shared SRAM is 8K in size. The PMC Shared SRAM (SSRAM) is mapped to Bus 0 Device 13 Function 3. The BAR field in that function points to the base address of the SSRAM. All addresses named "SSRAM Address" are relative to the base MMIO address. The Crashlog feature relies on about 5K of that region, though 1K of that is for the PMC trace buffer, which has other debug uses.

27.6.3 Description of Architectural Crashlog Components

- 1) Intel SoC components
 - a. Crash Data Requestor:
This functionality is the ability to generate requests to other subsystem components and either store or forward the data to a collector in the hierarchy. This would typically be the P-unit, PCU, TXE/ME, external baseboard controller (i.e. PMC)
 - b. Crash Data Detector:
The detector identifies when a subsystem has failed and data should be requested. Often this is the same device as the data requestor (i.e. PMC).
 - c. Crash Data Storage:
The functionality to provide storage for the crash data and persist across the system reset
 - d. Reset reason status register:
SOC register block that provides the System reset/wake/boot up reasons.
 - e. WDT timer:
Watchdog timer, with IO/MMIO/PCI based interfaces to AP to let the OS/IAFW to make use of this to recovery from system hard hang state. This timer expiry causes the system warm reset.
- 2) IAFW:
 - a. Crash Data Collector:
This entity which gathers all the Crash Data Requested and consolidates it (if needed) and publishes it to the operating system via standard interfaces
 - b. ACPI elements:
ACPI elements involved in reporting the crash records to the operating system e.g APCI APEI BERT table and reset reason reporting ACPI data structure called RSCI tables
 - c. Watchdog timer ACPI table:
Used to report the SOC WDT hardware resources to the operating system and report the WDT timer status across the boot.
- 3) OS level components:
 - a. OS Watchdog driver:
Consumes the IA Firmware reported "WDAT" ACPI table component to discover the SOC reported
 - b. Crashlog driver/application:
Consists of an OS provided or Intel provided driver/application that will take ACPI APEI BERT table, SMBIOS system information (need to know



what kind of Intel silicon is being used for debug purposes), ACPI RSCI table for how system was reset and got to this place in the execution of the system.

4) Intel Crashlog Decoder:

It will take the ACPI APEI BERT table and the other noted system info and decode into an ASCII readable text format file (in English only). The suggested naming convention for the extracted ACPI APEI BERT table data is as follows:

"IntelCrashlogData-YYYYMMDD-hhmmss.txt", where:

YYYY = current system year (decimal value)

MM = current system month (01-12)

DD = current system date (01-31)

hh = current system hour (24hour format – 00 -23)

mm = current system minutes (00-59)

ss = current system seconds (00-59)

5) Intel Crashlog file:

This file is the output of the Intel Crashlog Decoder and contains the following data:

- a. Header defining what files is and how it was generated.
- b. Data decoder from IntelRawCrashlog file in three additional sections:
 - APEI ACPI BERT tables decoded into subsystems such as PMC Crashlog, PUNIT Crashlog, CPU Crashlog, TXE Crashlog, and so on.

27.6.4 Crashlog debugging Tool

The Intel Crashlog Debugging Tool kit is available for customers using Apollo Lake based platforms. The tool can operate on a closed chassis platforms and does NOT require a connection to a debug probe such as Lauterbach.

Nevertheless, in case OS is not reachable, an open chassis data collection is also available via LTB connection. Refer Intel Crashlog Debugging tool on VIP #116448 to get more detailed information in the tool user guide

§



28 Security Engine

Intel® TXE 3.0 is the generation used in APL platform. TXE is an embedded microcontroller which runs firmware using host invisible resources. The firmware is digitally signed by Intel and is authenticated by TXE before it can be executed. TXE provides functionality even in the absence of an OS. This capability to function even when the OS is not present requires that TXE be able to run independently of the host.

Developers should note that CSE mentioning in the BIOS reference code refers to TXE.

28.1 Intel® TXE Overview

Refer TXE External Architecture Specification.

28.1.1 Intel® TXE Configuration

TXE needs to be configured before it can provide client functionalities. The configuration occurs at both the TXE platform level and the TXE feature level.

- **TXE Platform Level Configuration:** The configuration parameters at platform level impact the TXE platform as a whole and are independent of what features are currently supported on this platform. Another way to look at these parameters is that they impact the functionality and/or behavior of TXE kernel. Example of such configuration parameters is TXE enable/disable.
- **TXE Feature Level Configuration:** These configuration parameters impact individual TXE. These options are handled by the feature “module” and have no impact on the TXE kernel functionality/ behavior at all. An example of these options is Intel® Platform Trust Technology configuration options which are only applicable to Intel® Platform Trust Technology and do not impact any other feature.

28.2 Intel® TXE Platform Requirements

TXE firmware presents a virtual PCI device whose MMIO resource contains a ring buffer for message passing. The communication between IA FW and TXE firmware happens through messages formed according to Section [28.4](#). In addition, IA FW can determine the firmware status by reading the HECI device’s PCI registers during POST. The following sections detail the requirements that IA FW must follow.

28.2.1 Intel® TXE Firmware Disable

TXE firmware provides the following mechanisms to make TXE firmware run in disable/debug modes.

- **TXE Disable by IA FW.**
IA FW issues HECI command to disable TXE. Refer Section [28.4.3.11](#) for more detail. Once IA FW issue this disable command, TXE firmware will set HFS[19:16] “Operation Mode” field to “Soft Temporary Disable” after a global reset. It means the TXE firmware runs into the temporary disabled mode.



This command is used in the manufacturing and re-manufacturing environment, as well as for debug in the field.

- **TXE Debug Mode.**

The platform is able to stay in TXE debug mode by setting the variable SEC_DEBUG_MODE in TXE's FLASH region to 1 using the FLASH image creation tool. All the TXE functionalities such as firmware update will not be available in this mode. TXE only executes minimum firmware code required for platform boot, then TXE puts itself into the disable state.

In TXE debug mode, the TXE firmware sets HECI1_HFS[19:16] "Operation Mode" to "debug mode". Once firmware runs in Debug Mode, then it will not come out of this mode without re-flashing the whole new TXE image. The new TXE image needs to contain a SEC_DEBUG_MODE value of 0 that allows to bring the TXE out of TXE Debug Mode.

This option is used in the manufacturing environment.

- **TXE DnX Mode**

Beside above mechanisms, when system enters TXE DnX Mode, TXE FW will not be executed and can be updated via DnX. And IA FW will not be executed in this mode.

28.2.2 Intel® TXE Interfaces

TXE is exposed to the host as a multi-function PCI device. It supports three internal HECI functions, each with a complete PCI header.

- HECI1 D15:F0
- HECI2 D15:F1
- HECI3 D15:F2

IA FW interacts with TXE over HECI. For details on HECI message structure and programming guidelines, Refer Section [28.4](#).

The main HECI interface (HECI1) is a bi-directional fully asynchronous interface that passes messages between IA FW and TXE firmware. The second HECI interface (HECI2) is used to support IA FW SMI and SCI communication. The third HECI interface (HECI3) is targeted for future usage (Integrated Touch).

28.2.2.1 Intel® TXE Interface #1 Initialization

IA FW must program the TXE registers using the following sequence. IA FW should initialize HECI1 prior to the system memory initialization and set MSI as HECI Interrupt Deliver Mode. Then follow Section [28.3.3](#) to complete the HECI interface initialization before sending HECI message.

Table 28-1. HECI1 Initialization

Dev / Func	Offset	Field	Action
Dev 15 Func 0	10h	31:12	HECI MMIO Base Address Low. Program the MMIO BAR (lower 32 bits).
Dev 15 Func 0	14h	31:0	HECI MMIO Base Address High. Program the MMIO BAR (upper 32 bits).



Dev / Func	Offset	Field	Action
Dev 15 Func 0	04h	2:1	Memory Space Enable (MSE) : Set to enable TXE memory mapped register space. Bus Master Enable (BME) : Set to enable TXE bus master functionality.
Dev 15 Func 0	A0h	1:0	HECI Interrupt Delivery Mode (HIDM) 2'b00: Enable legacy or MSI interrupt

28.2.2.1.1 TXE Interface #1 Disable

HECI1 is used by IA FW and should not be disabled. In case HECI1 needs to be disabled for debugging purpose, IA FW must follow below sequence to disable HECI1 (D15:F0) after HECI1 initialization.

- Place HECI1 into D3 state in order to allow TXE FW to know this function is not operational. Refer Section [28.3.5](#) for the detail steps, including clear MSE and BME in PCI register offset 04h.
- Clear HECI MMIO Base Address High and HECI MMIO Base Address Low
- Disable HECI1 via bit 27 in FUNC_DIS_0 register of PMC GCR. Refer below table for the HECI1 disable register.

Table 28-2. HECI 1 Disable

Description	Register	/ Value
HECI1 Disable (Dev 15 Func 0)	PMC.GCR.FUNC_DIS_0.TXE_HECI1 PMC GCR Base + 0x1034[27]	1 = D15.F0 is disabled 0 = D15.F0 is enabled

28.2.2.2 Intel® TXE Interface #2 Initialization

IA FW must program the TXE registers using the following sequence. IA FW should initialize HECI2 and set SMI as HECI Interrupt Deliver Mode for Windows* OS, or MSI when Silent Lake enabled in Android* OS. Then follow Section [27.3.3](#) to complete the HECI interface initialization before sending HECI message.

Table 27-3. HECI2 Initialization

Dev / Func	Offset	Field	Action
Dev 15 Func 1	10h	31:12	HECI MMIO Base Address Low. Program the MMIO BAR (lower 32 bits).
Dev 15 Func 1	10h	31:0	HECI MMIO Base Address High. Program the MMIO BAR (upper 32 bits).
Dev 15 Func 1	04h	2:1	Memory Space Enable (MSE) : Set to enable TXE memory mapped register space. Bus Master Enable (BME) : Set to enable TXE bus master functionality.



Dev / Func	Offset	Field	Action
Dev 15 Func 1	A0h	1:0	HECI Interrupt Delivery Mode (HIDM) 2'b00: Enable legacy or MSI interrupt 2'b10: Enable SMI

Additional Requirement for Intel® TXE Interface #2

When HECI2 is used in Windows* OS with SMI, IA FW should move HECI2 into ACPI mode prior to Windows* OS loading. Moving HECI2 into ACPI mode (disable PCI register interface while keeping MMIO Space as a fixed resource) can be done by below PSF register.

And IA FW is expected to power manage HECI2, including moving HECI2 into D0i3 prior to Windows* OS loading. And from within SMM, IAFW must ensure HECI2 is out of D0i3, use the HECI2 and then put back the HECI2 to D0i3. Refer Section [28.3.5](#) and [28.3.6](#).

Note: Below register is only available on B-stepping or later stepping Apollo Lake SoC and derivatives. IA FW can keep HEC2 as PCI device for Windows* OS with A-Stepping SoC.

Table 28-4. HECI2 ACPI Mode

Description	PSF3 Register	Value
HECI2 ACPI Mode (Dev 15 Func 1)	PSF_3_AGNT_TO_SHDW_CFG_DIS_CSE_RSO_D15_F1 IOSF-SB Port C6h, Offset 1D38h, bit 0 NOTE: This register is not supported on A0 stepping SoC	1 = PCI Configuration Space is disabled 0 = PCI Configuration Space is enabled

Intel® TXE Interface #2 Disable

If HECI2 is not used, IA FW must follow below sequence to disable HECI2 (D15:F1) after HECI2 initialization.

- Place HECI2 into D3 state in order to allow TXE FW to know this function is not operational. Refer Section [28.3.5](#) for the detail steps, including clear MSE and BME in PCI register offset 04h.
- Clear HECI MMIO Base Address High and HECI MMIO Base Address Low
- Disable HECI2 via bit 26 in FUNC_DIS_0 register of PMC GCR. Refer below table for the HECI2 disable register.

Table 28-5. HECI2 Disable

Description	Register	/ Value
HECI2 Disable (Dev 15 Func 1)	PMC.GCR.FUNC_DIS_0.CSE_HECI2 PMC GCR Base + 0x1034[26]	1 = D15.F1 is disabled 0 = D15.F1 is enabled

28.2.2.3 TXE Interface #3 Initialization

Only in case when the platform plans to use HECL3, IA FW should initialize HECL3 and set MSI as HECL Interrupt Deliver Mode. Then follow Section [27.3.3](#) to complete the HECL interface initialization before sending HECL message.

Table 28-6. HECL3 Initialization

Dev / Func	Offset	Field	Action
Dev 15 Func 2	10h	31:12	HECL MMIO Base Address Low. Program the MMIO BAR (lower 32 bits).
Dev 15 Func 2	10h	31:0	HECL MMIO Base Address High. Program the MMIO BAR (upper 32 bits).
Dev 15 Func 2	04h	2:1	Memory Space Enable (MSE): Set to enable TXE memory mapped register space. Bus Master Enable (BME): Set to enable TXE bus master functionality.
Dev 15 Func 2	A0h	1:0	HECL Interrupt Delivery Mode (HIDM) 2'b00: Enable legacy or MSI interrupt

28.2.2.3.1 TXE Interface #3 Disable

If HECL3 is not used, IA FW must follow below sequence to disable HECL3 (D15:F2) after HECL3 initialization.

- Place HECL3 into D3 state in order to allow TXE FW to know this function is not operational. Refer Section [28.3.5](#) for the detail steps, including clear MSE and BME in PCI register offset 04h.
- Clear HECL MMIO Base Address High and HECL MMIO Base Address Low
- Disable HECL3 via bit 26 in FUNC_DIS_0 register of PMC GCR. Refer below table for the HECL3 disable register.

Table 28-7. HECL3 Disable

Description	Register	Bit/ Value
HECL3 Disable (Dev 15 Func 2)	PMC.GCR.FUNC_DIS_0.CSE_HECL3 PMC GCR Base + 0x1034[25]	1 = D15.F2 is disabled 0 = D15.F2 is enabled

28.2.3 Intel® TXE – Host Firmware Status Register

The host visible TXE PCI device at Device 15 Function 0 (HECL1) has six 32-bit host firmware status registers (HFS) located in the following PCI configuration space. The TXE firmware will write status information about the state of the TXE firmware into these registers. IA FW can read these registers to determine the current state of the firmware. The definitions of these six 32-bit host firmware status registers are described in below table.

Table 28-8. TXE Firmware Status Registers

Dev / Func	Offset	Register Name	Description
Dev 15 Func 0	40h	HECI1_HFS	FS_HA TXE Firmware Status Host Access Shadow of FWSTS1 in TXE FW
Dev 15 Func 0	48h	HECI1_GS_SHDW1	GSS1 TXE General Status Shadow 1 Shadow of FWSTS2 in TXE FW
Dev 15 Func 0	60h	HECI1_GS_SHDW2	GSS2 TXE General Status Shadow 2 Shadow of FWSTS3 in TXE FW
Dev 15 Func 0	64h	HECI1_GS_SHDW3	GSS3 TXE General Status Shadow 3 Shadow of FWSTS4 in TXE FW
Dev 15 Func 0	68h	HECI1_GS_SHDW4	GSS4 TXE General Status Shadow 4 Shadow of FWSTS5 in TXE FW
Dev 15 Func 0	6Ch	HECI1_GS_SHDW5	GSS5 TXE General Status Shadow 5 Shadow of FWSTS6 in TXE FW

Table 28-9. Host Firmware Status Register (HECI1_HFS) Definition (Offset 40h)

Bits	Description
3:0	TXE Current Working State: This field describes the current working state of the TXE firmware. In spoken language, the "Current Working State" indicates what firmware is doing at this moment. Further information about the progress within the Current Working State can be found in the Extended Status Data field of this register. The encoding of this field can be found in Table 28-15 .
4	Manufacturing Mode: When this bit is set, the TXE platform is still in Manufacturing mode. Host can use this bit to inform user that the platform is NOT ready for production yet. This bit is set as long as TXE Manufacturing Mode Done bit is set.
5	FPT Bad: This bit is set when the firmware discovers a bad checksum of Flash Partition Table (FPT). When this bit set, it may or may not have the error code shown in bits [15:12]. The system can get this bit clear by flashing the proper firmware image again.
8:6	TXE Current Operation State: This field describes the state that TXE is currently functioning in at this moment. It's the combination of TXE power state and UMA. The "Current Operation State" is set only upon entering the true hardware state, e.g. We set it to M0 only after PLL's are locked to M0 freq and controller is set to SD. 000 – Preboot (Default) 001 – M0 with UMA 010 – M0 Power Gated 011 – Reserved 100 – Reserved

Bits	Description
	101 – M0 without UMA 110 – Bring up 111 – M0 without UMA but with error
9	FWInitComplete: When this bit is clear, TXE firmware is still in loading process and has not made it to final steady state yet. When TXE firmware has fully entered a stable state, this bit is set to 1 and "Current Working State" field of this register provides the steady state of TXE subsystem.
10	FT BUP LD FLR: This bit is set when TXE firmware is not able to load BRINGUP from the fault tolerant (FT) code. When this bit set, it may or may not have the error code shown in bit [15:12]. The reason is because the firmware can load BRINGUP from the non-fault tolerant (NFT) code
11	FW UPD In Progress: This bit is set when data migration is required during the TXE firmware update process.
15:12	Error Code: Set to a nonzero value if the TXE firmware has encountered a fatal error. The TXE firmware has stopped normal operation. Further information about the error can be found in the Extended Status Data field of this register. The encoding of this field can be found in Table 28-16 .
19:16	TXE Current Operation Mode: The field indicates the TXE firmware mode of operation during this boot cycle. The host must look at this field only after DRAM_INIT_DONE (DID) register based message unless the firmware is in error before sending the DID. The encoding of this field can be found in Table 28-17 . The Operation Mode does not change at runtime.
20:23	Reserved
24	Reserved
27:25	ACK Data
31:28	IA FW MSG ACK

Table 28-10. General Status Shadow #1 (HECI1_GS_SHDW1) Definition (Offset 48h)

Bits	Description
0	BIST In Progress: If this bit is set, TXE firmware BIST test is in progress. BIST will happen at the first success ME boot. TXE BIST test is reset tolerant, any reset coming from IA FW/ host would not impact the BIST. TXE firmware will continue the test in the next reset cycle until it get done.
2:1	Reserved
3	Reserved
4	Reserved
5	Reserved
6	MFS Failure: If MFS Failure bit is set, TXE firmware detects a TXE File System corruption. The real time OS can use this bit to indicate the MFS failure problem.
7	Reserved
8	Reserved
9	Reserved



Bits	Description
10	TXE Power Gating Indicator: This bit indicated if TXE is entering Power Gating. 1 – power gated, 0 – not power gated
12:11	Reserved
13	Reserved
14:15	Reserved
27:16	Extended Status Data: These bits provided extended status data for the current state of operation of the firmware. Or, if the firmware is in a fatal error state, these bits provide extended error code information. The encoding of this field can be found in Table 28-18 .
31:28	Infrastructure Progress Code: This field identifies the infrastructure progress code. For example, when firmware comes out of reset, this field would identify the infrastructure moving from ROM to BRINGUP to Micro Kernel to Policy module, and so on. Note: This field is specific to infrastructure and does not represent the entire firmware. Refer the Working State and Error Code fields to identify current state of the firmware. The encoding of this field can be found in Table 28-19 .

Table 28-11. General Status Shadow #2 (HECI1_GS_SHDW2) Definition (Offset 60h)

Bits	Description
0	CHO: Chunk 0
1	CH1: Chunk 1
2	CH2: Chunk 2
3	CH3: Chunk 3
8:4	Reserved
9	IVR: IBB Verification Result
10	IVD: IBB Verification Done
13:11	Reserved
27:14	AIS: Actual IBB size (1KB Granularity)
29:28	NOC: Number of chunks (0=reserved, 1=1chunk of 128KB, 2=2chunks of 64KB, 3=4chunks of 32KB)
31:30	Reserved

Table 28-12. General Status Shadow #3 (HECI1_GS_SHDW3) Definition (Offset 64h)

Bits	Description
9:0	Reserved
10	Reserved
11	Reserved



Bits	Description
12	All TPMs Disconnected: If this bit is set, TXE firmware BUP has executed TPM disconnection flow.
13	Reserved
14	TXE Verified Boot FWSTS Valid: If this bit is set, TXE Verified Boot related FWSTS bits are valid to read.
15	TXE Verified Boot Self Test: If this bit is set, TXE Verified Boot Self Test resulted in an error during TXE firmware BUP.
31:16	Reserved

Table 28-13. General Status Shadow #4 (HECI1_GS_SHDW4) Definition (Offset 68h)

Bits	Description
31:0	Reserved

Table 28-14. General Status Shadow #5 (HECI1_GS_SHDW5) Definition (Offset 6Ch)

Bits	Description
31:0	Reserved

Table 28-15. Host Firmware Status Register (HECI1_HFS)– Bits [3:0] Current Working State Values

Value	Description
0	Reset: The TXE firmware is in a reset state. The firmware will exit this state within 1 millisecond.
1	Initializing: The TXE firmware is initializing, the final functional state is NOT known at this time. The firmware will exit this state within 4 seconds.
2	Recovery: The TXE firmware is recovering from an image update failure. Upon completion of this state the TXE firmware will enter the Initializing State. The firmware will stay in recovery state until a new firmware update is run.
3-4	Reserved
5	Normal: The TXE firmware is operating in normal state.
6	Reserved
7	OP State Transition: TXE firmware sets this state before starting a transition to a new operation state.
8	Reserved

**Table 28-16. Host Firmware Status Register (HECI1_HFS)– Bits [15:12] Error Code Values**

Value	Description
0	No Error: Firmware is operational in current state.
1	Uncategorized Failure: The TXE firmware has experienced an uncategorized error and has disabled itself. Further details of the failure can be found in the Extended Status Data field of the Firmware Status Register. This state can only be exited by putting the TXE subsystem in Moff, which may be achieved by a power button override (user holding down the power button for 5 seconds).
2	Reserved
3	Image Failure: The TXE firmware stored in the system flash is not valid. If supported by the platform, the host should program a full firmware image into TXE flash region through the SPI controller. After completion of the TXE flash region update, the host should generate a full system reset of the platform (global reset). In addition, Image Failure results in a platform involuntary 30-minute shut down triggered by TXE. IA FW shall post a warning message as part of the error handling flow.
4	Debug Failure: Set when more data is available to the firmware on the cause of the failure. Refer the Extended Status Data field for more details.

Table 28-17. Host Firmware Status Register (HECI1_HFS)– Bits [19:16] Operation Mode Values

Value	Description
0	Normal: The TXE is running in Normal mode
1	Reserved
2	DEBUG MODE: The TXE firmware enters this operation mode when the softstrap PCHSTRP10 bit [7] is 1. Firmware stays in this mode until the softstrap PCHSTRP10 bit [7] set to 0.
3	Soft Temporary Disable: The TXE firmware enters this operation mode when the IA FW sends the Temporary Disable message to firmware. Firmware stays in this mode until the IA FW sends a follow-up message to get TXE out of this mode.
4	Reserved
5	Reserved

Table 28-18. General Status Shadow #1 (HECI1_GS_SHDW1)– Bits [27:16] Extended Status Data

Bits	Description
23:16	Current State: status of current operational module.
27:24	Current PmEvent: status of current TXE power management event 0x0: Clean Moff->Mx wake 0x1: Moff->Mx wake after an error 0x2: Clean global reset 0x3: Global reset after an error 0x4: Clean TXE reset

Bits	Description
	0x5: TXE reset due to exception 0x6: Pseudo-global reset 0x7: S0/M0->Sx/M3 0x8: Sx/M3->S0/M0 0x9: Non-power cycle reset 0xa: Power cycle reset through M3 0xb: Power cycle reset through Moff 0xc: Sx/Mx->Sx/Moff

Table 28-19. General Status Shadow #1 – Bits [31:28] Infrastructure Progress Code Values

Value	Description
0	ROM: The TXE is in ROM phase.
1	BUP: The TXE is in BRINGUP phase.
2	uKernel: The TXE is in Micro Kernel phase.
3	Policy Module: The TXE is in Policy Module phase.
4	Module Loading: The TXE is starting to load modules in M0 or M3 Operation Mode.
6	Host Communication: The TXE is up and running. Host communication established

Table 28-20. General Status Shadow #4 (HECI 1_GS_SHDW4) – Bits [7:3] Error Status Code (ECS)

Value	Description
0x01	Boot Guard initialization failed
0x02	KM (key manifest) verification failed
0x03	BPM (boot policy manifest) verification failed
0x04	IBB (initial boot block) verification failed
0x05	FIT (firmware interface table) processing failed
0x06	DMA (direct memory access) setup failed
0x07	NEM (non-eviction mode) setup failed
0x08	TPM (trusted platform module) startup failed
0x09	IBB (initial boot block) measurement failed
0x0A	TXE connection failed

28.2.4 IA FW Boot Paths

The following IA FW boot flows should be implemented in the IA FW.

HECI 1_HFS Current Working State [bits 3:0]



- If '0x02' (Recovery), then the IA FW takes TXE Recovery IA FW path. Refer Section [28.2.4.3](#).
- If '0x05' (Normal), then the IA FW takes the normal firmware IA FW path.

HECI1_HFS FPT Bad [bit 5]

- If '0x01' (fatal firmware error), then the IA FW takes TXE Error IA FW path. Refer Section [28.2.4.2](#).

HECI1_HFS Error Code [bits 15:12]

- If '0x03' (firmware image is not found), then the IA FW takes TXE Error IA FW path without DID message. Refer Section [28.2.4.1](#). **In addition, Image Failure results in a platform involuntary 30-minute shut down triggered by TXE. IA FW shall post the warning message as part of the error handling flow.**
- If any other 'non-zero' value, then the IA FW takes TXE Error IA FW path. Refer Section [28.2.4.2](#).

HECI1_HFS Operation Mode [bits 19:16]

- If '0x00' (Normal), then the IA FW takes the normal firmware IA FW path.
- If '0x02' (Debug Mode) or '0x03' (Soft Temporary Disable), then the IA FW takes the TXE Disable IA FW path. Refer Section [28.2.4.4](#).

HECI1_HFS FW UPD In Progress [bit 11]

- If '0x01' (firmware is in update process), then the IA FW takes the TXE Firmware Update IA FW Path. Refer Section [28.2.4.5](#).

HECI1_GS_SHDW1 BIST In Progress [bit 0]

- In the first TXE firmware boot after flash, the auto BIST will not start its test until HECI1_HFS FWInitComplete [bit 9] bit is set. After HECI1_HFS FWInitComplete [bit 9] is set to 1, the TXE firmware will execute the BIST test. At this point, the TXE firmware will set the HECI1_GS_SHDW1 BIST In Progress [bit 0] to be 1. In the beginning of BIST test, the testing model will trigger a GRST from TXE firmware side.
- TXE firmware will set the HECI1_GS_SHDW1 BIST In Progress [bit 0] to be 0 after completing BIST test.

28.2.4.1 TXE Error Without DID Message IA FW Path

Task	Description
HECI Message	The IA FW does not send any HECI messages including DID and EOP message.
HECI Devices	HECI1 device should be enabled by the IA FW so it allows the user to update the new firmware and take the firmware out of a recovery or error condition. Hide HECI2 and HECI3.

28.2.4.2 TXE Error IA FW Path

Task	Description
HECI Message	The IA FW does not send any HECL messages except for the DRAM Init Done message. Moreover, the IA FW does not even send EOP message
HECI Devices	HECI1 device should be enabled by the IA FW so it allows the user to update the new firmware and take the firmware out of a recovery or error condition. Hide HECL2 and HECL3.

28.2.4.3 TXE Recovery IA FW Path

Task	Description
HECI Message	The IA FW does not send any HECL messages except for the DRAM Init Done message . Moreover, the IA FW does not even send EOP message
HECI Devices	HECI1 device should be enabled by the IA FW so it allows the user to update the new firmware and make the firmware out of from recovery or error condition. Hide HECL2 and HECL3.

28.2.4.4 TXE Disable IA FW Path

HECI1_HFS Operation Mode [bits 19:16] = 0x02 (Debug Mode)

Task	Description
HECI Message	The IA FW does not send any HECL messages including DRAM Init Done (DID) and EOP message.
HECI Devices	Hide HECL1, HECL2 and HECL3 before OS boot. It means there is no HECL drivers loaded in OS environment

HECI1_HFS Operation Mode [bits 19:16] = 0x03 (Software Temporary Disable)

Task	Description
HECI Message	The IA FW does not send any HECL messages except for the DRAM Init Done (DID) message. Moreover, the IA FW does not even send EOP message
HECI Devices	Hide HECL1, HECL2 and HECL3 before OS boot. It means there is no HECL drivers loaded in OS environment

28.2.4.5 TXE Firmware Update IA FW Path

The following procedure is executed before sending the EOP.

The firmware has to run in HECL1_HFS BITS [3:0] normal state and BITS [19:16] normal mode then the IA FW reads the HECL1_HFS bit [11] "FW UPD in Progress" value. If the value is 1, IA FW starts a 90-seconds loop to wait for the firmware to clear the bit to 0.



The recommended message that is showed by the IA FW during the waiting period of time is – “Intel® firmware update processes is in progress. It may take up to 90 seconds... Please wait....”

Once the IA FW completes the waiting loop, the IA FW sends EOP and continues the boot.

28.2.5 TXE IA FW Requirements

IA FW must perform certain specific steps and requirements within the IA FW POST to ensure correct TXE functionality. The following list mentions some possible steps in IA FW POST in which new requirements for TXE must be satisfied.

- IA FW must initialize TXE interface PCI resource registers before it can send/receive any HECL messages. Refer Section [28.2.2](#).
- After system memory initialization, the IA FW must send DRAM Init Done (DID) HECL message to TXE. Refer Section [28.2.5.2](#) for details.
- IA FW must program the interrupt controller such that TXE interrupts to host are “level”.
- Perform additional steps and send any appropriate TXE messages at the following times:
 - When triggering a system reset during IA FW POST. Refer Section [28.2.5.1](#) for details.
 - IA FW must also program SSID/SSID for Bus 0 Device 15 Function 0, 1 and 2 to non-zero values (specific to OEM motherboard). Failure to program these to non-zero values is a violation of the PCI specification.
 - At the end of IA FW POST, IA FW must send an “END OF POST” HECL message to TXE declaring end of POST and start of OS load. IA FW must also wait for an “END of POST” response message from TXE before proceeding.
- HECL2 should be moved into Link-Down mode after HECL2 initialization, and host init HECL2 Link-up (SetHeciActive) at before SMM HECL2 msg and Link-down (SetHeci2lDel) after all SMM HECL msg are completed.
- IA FW should put HECL1 and HECL3 into D0i3 after EOS.

28.2.5.1 System Global Reset on TXE Enabled System

During POST, IA FW may need to trigger a global reset as part of platform initialization or for error handling. In a TXE enabled platform, TXE is running in parallel to the host system.

IA FW has 2 ways to perform global reset (reset both host, TXE, and PMC). One is register-based (through 0Xcf9 write of 06h or 0Eh command with register PBASE + 0x4C bit 20 CF9GR set). Another is to issue the Global Reset HECL message. Because any attempts to cause global reset without synchronizing the two sides might cause unwanted side effects, such as unwritten flash data that will get destroyed if the host were to cause a global reset without informing TXE, the recommended method is to send a Global Reset HECL message when the following conditions were met:

- IA FW just needs to complete the TXE HECL interface initialization and check register HECL1_HFS – if the register is accessible and TXE is not in ERROR state and



is accepting HECI commands, then IA FW should be able to use Global Reset HECI message to trigger global reset.

- IA FW should be able to use Global Reset HECI message prior to End of Post HECI message. After End of Post HECI message, TXE firmware will not accept Global Reset HECI message any more.

Furthermore, if TXE runs in ERROR state, IA FW can use the above mentioned register-based method of issuing of Global Reset.

IA FW must also ensure that Global Reset is disabled and locked (via register PBASE + 0x4C bits 20 and 31) before handing control to the OS in order to prevent the host from issuing Global Resets.

28.2.5.2 System Memory Initialization on TXE Enabled System

Prior to system memory initialization, IA FW should verify that TXE is not disabled (Refer Section [28.2.1](#)).

Currently, the maximum memory size for TXE usage is 64MB. MRC shall assume the maximum for TXE before sending DRAM Init Done HECI message method below and allocate the requested amount of memory into the system memory map based on the result buffer of DRAM Init Done HECI message. The MRC must allocate TXE UMA below TOM and not lower than 1GB memory address (0x10000000), and there are no other restrictions on where TXE UMA can be located.

After system memory initialization is complete, the IA FW saves the data that is passed out from the MRC to NVRAM.

Note: The MRC implementation must be such that if a warm reset is issued at any time between the DRAM Init Done ACK (Refer in steps below) and the point at which MRC S3 data is saved to NVRAM, MRC shall issue a cold (power cycle) reset on the next boot, ensuring that valid memory configuration settings will be applied.

If TXE is not disabled, the IA FW must send the DRAM Init Done HECI message indicating to TXE firmware that DRAM initialization is complete and TXE UMA is ready for use. DRAM Init Done HECI message utilizes MKHI message format (Refer Table 3-20. DRAM Init Done MKHI Message for more details on this message). The DRAM Init Done MKHI message should be sent by the IA FW on all boot flows.

After the IA FW sends the DRAM Init Done HECI message, IA FW should wait for DRAM Init Done ACK MKHI message from TXE. This ACK message will include a requested "IA FW Action", which the IA FW should act on the "IA FW Action" as soon as possible upon receiving the DRAM Init Done ACK HECI message. Section [27.4.5](#) describes the format of the DRAM Init Done ACK HECI message.

As soon as the DRAM Init Done ACK HECI is, IA FW should continue as instructed, according to the requested TXE IA FW Action specified by the **ACK Data**.

If the IA FW needs to perform any power transition after system powers on, the IA FW **must** wait until receiving the DRAM Init Done ACK HECI message. The IA FW must conduct the power transition based on the "IA FW Action" request. If the "IA FW Action" instructs that IA FW can continue to POST, then the IA FW can apply its own power transition flow after that.



28.2.5.3 Data Clear on TXE Enabled System

Background and Purpose:

TXE data region holds critical manufacturing configuration done by OEM/ODM at manufacturing time + end user data when user uses system (e.g. fTPM data, content protection data, DAL data, etc.). TXE data clear flow is designed for customers to use during manufacturing / refurbishing scenarios. TXE Data clear API has been provided to OEM/ODM factory flow in order to clear out all the manufacturing configuration data to clean the TXE filesystem, return system to pre-production stage, then reconfigure system (i.e. refurbish / re-purpose). TXE data clear flow has been designed for OEM/ODM to support system refurbish / re-purposing scenarios only. After data clear flow has been executed, OEM/ODM is required to re-perform the appropriate manufacturing configurations then re-ship to end user.

Usage:

Use during manufacturing / refurbishing scenarios.

Do not use for systems at the end-user possession (i.e. in-field push/pull updates), will be a significant risk to system stability.

Impact when system executed Data Clear Flow:

Data clear flow clears all TXE & BIOS data stored in device expansion region on SPI boot devices. The immediate next boot after data clear flow will be system first boot scenario where TXE will re-build the filesystem in device expansion and expect manufacturer to reconfigure its required manufacturing steps.

The following areas are impacted once data clear is issued:

1. Any PTT/fTPM information would be cleared.
2. Any BIOS settings set by the end customer would be lost if they are using TXE file system to store UEFI variables.
 - a. All BIOS setting during ODM production will also be lost or restored to default as well including boot source, BIOS lock/FPRR protection, BIOS PTT enable/disable and so on.
3. Any manufacturing configuration done by FPT tool for NVAR updates during OEM/ODM factory will be lost
 - a. NVARs configurations will be cleared (e.g. Enable disable of: DAL, PAVP, PTT, FWU OEM_ID, DnX Cable detection timeout and so on.)
4. ISH configuration/provisioning data will be lost
5. EOM (End of Manufacturing) variable will become cleared/"not set" and system will behave as pre-EOM stage (R&D/development/pre-production phase)
6. Debug file of HW FPF status will be lost (This file is saved after FPFs are committed)
7. TXE will be in manufacturing mode where OEM debug hooks are open to access without protection
8. Platform power consumption may also be impact since system is in pre-production phase.

Impact if data clear is used for end-user:

If data clear is performed for in-field updates, all systems accepting such updates in the field will be required to be shipped to OEM/ODM factory to recover then re-ship to end-users.

Call for Action:

Make sure that TXE data clear is not performed for updates that are pushed to end-user systems in-field. Use firmware/BIOS updates that include data clear for systems during manufacturing only. Once data clear is issued, ensure to re-perform the required manufacturing steps to make the system shippable to end-users.

28.3

HECI Interface and HECL Message

This section describes the aspect of the runtime operation of the HECL interface, including how IA FW will use the HECL hardware interface registers and circular buffers.

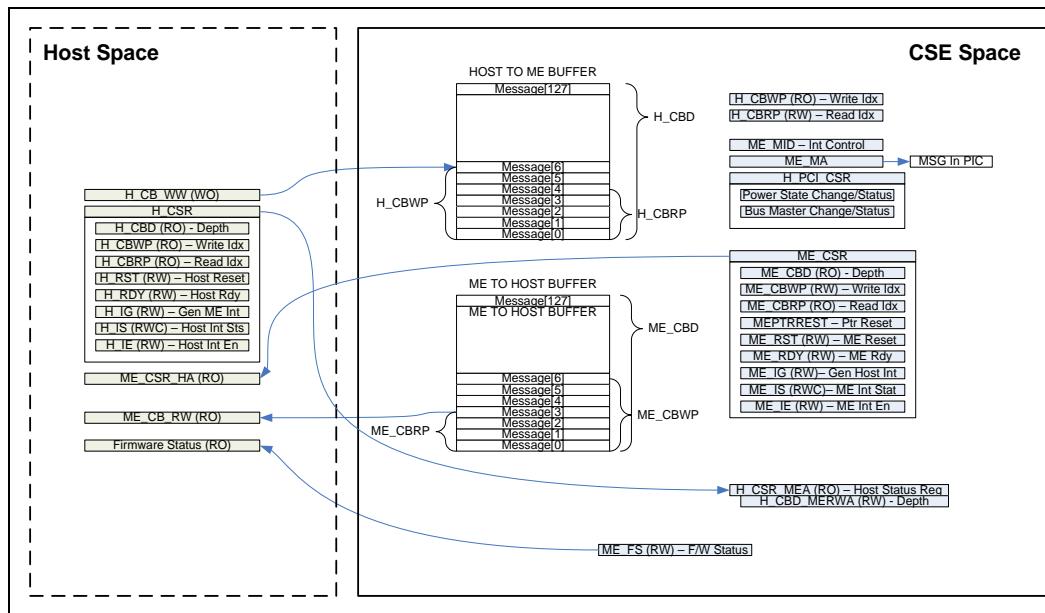
A general message format is used to pass data through HECL circular buffers. The same message format is used for messages from the host (IA FW or Software) to TXE (TXE FW) and from TXE to the host. The message consists of a generic header and a data payload. The format and contents of the data payload are opaque to the host and firmware HECL drivers, and are only interpreted by the associated clients in the host and firmware domains.

28.3.1

TXE Circular Buffer

IA FW and TXE firmware exchange messages using a set of circular buffers and Control/Status registers (CSR's). There are two circular buffers (CBs), one for passing messages from the host to the TXE and one for passing messages from the TXE to the host. The messages may be of variable length, but must consist of one or more 32-bit DWords; message size info will be part of message. The HECL hardware interface provides CB Read and Write Pointers and CB Depth fields for software / firmware to manage the circular buffers. CB Full and CB Empty conditions can be calculated by FW/SW from the CB Read and Write pointer and CB Depth information.

Figure 28-1. Basic Diagram of HECL Interface



28.3.2 Circular Buffers and Control/Status Registers

All the HECL Interfaces (HECL1, HECL2, HECL3) have their own circular buffers and control/status registers (CSR) according to the HECL MMIO Base Address programmed during HECL interface initialization mentioned in Section 28.2.2. And definitions of register in HECL MMIO are the same for each HECL interface.

Host access to HECL MMIO Base is in DWord transactions.

Table 28-21. HECL1 MMIO Registers

Offset	Default	Name	Description
0x0	0000_0000h	HECL1_H_CB_WW	Host CB Write Window
0x4	8000_0000h	HECL1_H_CSR	Host Control and Status Register
0x8	FFFF_FFFFh	HECL1_CSE_CB_RW	TXE Circular Buffer Read Window
0xC	8000_0000h	HECL1_CSE_CSR_HA	TXE Control and Status Register Host Access
0x800	0000_0000h	HECL1_D0I3C	D0I3 Control

Table 28-22. HECL2 MMIO Registers

Offset	Default	Name	Description
0x0	0000_0000h	HECL2_H_CB_WW	Host CB Write Window
0x4	8000_0000h	HECL2_H_CSR	Host Control and Status Register
0x8	FFFF_FFFFh	HECL2_CSE_CB_RW	TXE Circular Buffer Read Window

Offset	Default	Name	Description
0xC	8000_0000h	HECI2_CSE_CSR_HA	TXE Control and Status Register Host Access
0x800	0000_0000h	HECI2_D0I3C	D0I3 Control

Table 28-23. HECI3 MMIO Registers

Offset	Default	Name	Description
0x0	0000_0000h	HECI3_H_CB_WW	Host CB Write Window
0x4	8000_0000h	HECI3_H_CSR	Host Control and Status Register
0x8	FFFF_FFFFh	HECI3_CSE_CB_RW	TXE Circular Buffer Read Window
0xC	8000_0000h	HECI3_CSE_CSR_HA	TXE Control and Status Register Host Access
0x800	0000_0000h	HECI3_D0I3C	D0I3 Control

Table 28-24. Host CB Write Window (HECI1_H_CB_WW)

Bits	Access Type	Default	Description
31:0	WO	32'h00000000	Host Circular Buffer Write Window Field (H_CB_WWF) This field is for host to write into its circular buffer. This field is write only, reads will return arbitrary data. Writes to this register will increment the H_CBWP as long as CSE_RDY is 1. When CSE_RDY is 0, writes to this register have no effect and are not delivered to the H_CB, nor is H_CBWP incremented.

Table 28-25. Host Control and Status Register (HECI1_H_CSR)

Bits	Access Type	Default	Description
31:24	RO	8'h80	Host Circular Buffer Depth (H_CBD) This field indicates the maximum number of 32 bit entries available in the host circular buffer (H_CB). IA FW uses this field along with the H_CBRP and H_CBWP fields to calculate the number of valid entries in the H_CB to read or number of entries available for write. This field is hardwired to represent a depth of 128 entries.
23:16	RO/V	8'h00	Host CB Write Pointer (H_CBWP) Points to next location in the H_CB for host to write the data. Software uses this field along with H_CBRP and H_CBD fields to calculate the number of valid entries in the H_CB to read or number of entries available for write.
15:8	RO/V	8'h00	Host CB Read Pointer (H_CBRP)

Bits	Access Type	Default	Description
			Points to next location in the H_CB where a valid data is available for embedded controller to read. Software uses this field along with H_CBWR and H_CBD fields to calculate the number of valid entries in the host CB to read or number of entries available for write.
7	RO	1'b0	Reserved
6	RW/1C/V	1'b0	Host DOI 3C Interrupt Status (H_DOI 3C_IS) HW sets this bit to 1 when DOI3C.IR is set and DOI3C.CIP transitions from 1 to 0. Host clears this bit to 0 by writing a 1 to this bit position. H_DOI3C_IE has no effect on this bit.
5	RW	1'b0	Host DOI 3C Interrupt Enable (H_DOI 3C_IE) Host sets this bit to 1 to enable the host interrupt (MSI, INTx, SMI or SCI) to be asserted when H_DOI3C_IS is set to 1.
4	RW	1'b0	Host Reset (H_RST) Setting this bit to 1 will initiate the HECI reset sequence to get the circular buffers into a known good state for host and TXE communication. When this bit transitions from 0 to 1, hardware will clear the H_RDY and CSE_RDY bits.
3	RW/V	1'b0	Host Ready (H_RDY) This bit indicates that the host is ready to process messages.
2	RW/V	1'b0	Host Interrupt Generate (H_IG) Once message(s) are written into its CB, the host sets this bit to one for the HW to set the CSE_IS bit in the CSE_CSR and to generate an interrupt message to TXE. HW then clears this bit to 0.
1	RW/1C/V	1'b0	Host Interrupt Status (H_IS) HW sets this bit to 1 when CSE_IG bit is set to 1. Host clears this bit to 0 by writing a 1 to this bit position. H_IE has no effect on this bit.
0	RW	1'b0	Host Interrupt Enable (H_IE) Host sets this bit to 1 to enable the host interrupt (MSI, INTx, SMI or SCI) to be asserted when H_IS is set to 1.

Table 28-26. TXE Circular Buffer Read Window (HECI 1_CSE_CB_RW)

Bits	Access Type	Default	Description
31:0	RO	32'hFFFFFF	TXE Circular Buffer Read Window Field (CSE_CB_RWF) This bit field is for host to read from the TXE Circular Buffer. This field is read only, writes have no effect.

			Reads to this register will increment the CSE_CBRP as long as CSE_RDY is 1. When CSE_RDY is 0, reads to this register have no effect, all 1s are returned, and CSE_CBRP is not incremented.
--	--	--	---

Table 28-27. TXE Control and Status Register Host Access (HECI1_CSE_CSR_HA)

Bits	Access Type	Default	Description
31:24	RO	8'h80	TXE Circular Buffer Depth Host Read Access (CSE_CBD_HRA) Host read access to CSE_CBD.
23:16	RO/V	8'h00	TXE CB Write Pointer Host Read Access (CSE_CBWP_HRA) Host read only access to CSE_CBWP.
15:8	RO/V	8'h00	TXE CB Read Pointer Host Read Access (CSE_CBRP_HRA) Host read only access to CSE_CBRP.
7	RO/V	1'b0	NMI Status Host Read Access (NMI_STS_HRA) Host read access to NMI_STS.
6	RO	1'b0	Reserved
5	RO/V	1'b0	Pointer Reset Host Read Access (PTR_RST_HRA) Host read access to PTR_RST.
4	RO/V	1'b0	TXE Reset Host Read Access (CSE_RST_HRA) Host read access to CSE_RST.
3	RO/V	1'b0	TXE Ready Host Read Access (CSE_RDY_HRA) Host read access to CSE_RDY.
2	RO/V	1'b0	TXE Interrupt Generate Host Read Access (CSE_IG_HRA) Host read only access to CSE_IG
1	RO/V	1'b0	TXE Interrupt Status Host Read Access (CSE_IS_HRA) Host read only access to CSE_IS.
0	RO/V	1'b0	TXE Interrupt Enable Host Read Access (CSE_IE_HRA) Host read only access to CSE_IE.

Table 28-28. DOI3C Control (HECI1_DOI3C)

Bits	Access Type	Default	Description
31:5	RO	28'h0	Reserved
4	RO	1'b1	Interrupt Request Capable (IRC)
3	RO	1'b0	Restore Required (RR)
2	RW	1'b0	DOI3 (DOI3C)

Bits	Access Type	Default	Description
			<p>SW sets this bit to 1 in order to move the IP into the D0i3 state. Clearing this bit will return the IP into the fully active D0 state.</p> <p>When this bit changes state, TXE may be interrupted (Refer H_PCI_CSR.D0I3C_IS register's description)</p>
1	RW	1'b0	<p>Interrupt Request (IR)</p> <p>SW sets this bit to 1 to ask for an interrupt to be generated on completion of the command. SW must clear or set this bit on each write to this register.</p> <p>When this bit is set to 1, Command-in-Progress deassertion is captured in H_CSR.H_D0I3C_IS. If H_CSR.H_D0I3C_IE is 1 as well, host interrupt will be initiated.</p>
0	RO/V	1'b0	<p>Command-in-Progress (CIP)</p> <p>HW sets this bit on a 0->1 or 1->0 transition of D0i3.</p> <p>While set, the other bits in this register are not valid and it is illegal for SW to write to any of them.</p> <p>HW clears this bit when TXE FW clears its own D0i3 interrupt status bit indicating completion of the D0i3 transition command.</p> <p>While clear, all other bits in this register are valid and SW may write to them.</p> <p>If Interrupt Request (IR) was set for the current command, HW may clear this bit before the interrupt has been made visible to SW, since when SW actually handles a particular interrupt is not visible to HW.</p> <p>SW writes to this bit have no effect.</p>

28.3.3 HECI Interface Initialization and Reset

The following sections describe the reset flows of the HECL interface to get the HECL interface to a known state from any unknown state and establish a starting point for HECL message communication. These reset flows describes an interface reset sequence performed by IA FW and TXE firmware cooperation, and does not Refer a hardware reset sequence such as host partition reset or TXE partition reset.

IA FW should also initial this reset flow when it observes CSE_RST_HRA (HECL MMIO 0xC[4]) is 1 in CSE_CRS_HA when receiving interrupt from TXE.

IA FW should also reset HECL interface if it receives a malformed or no response message from TXE firmware. IA FW should retry a command at least one time (up to two three; total 3 attempts) if IA FW receives malformed or no response. If IA FW again detects a failure then it should take TXE Error path.

IA FW has loaded and desires to get the HECL message buffers in a known good state prior to initiating message transfers. IA FW must perform this sequence after host partition reset is de-asserted and before sending any messages through the HECL interface. IA FW may also perform this sequence any time messages through HECL have become out of sync or the interface is in an unknown state.

1. Perform PCI initialization (only needed after host partition reset)
2. Host terminates its own reads and writes to HECL, will not initiate any until this sequence completes. (Not needed after host partition reset).



3. Host writes to the H_CSR register, setting both the H_RST bit (HECI MMIO 0x4[4]) and the H_IG bit (HECI MMIO 0x4[2]) to 1. IA FW may also set H_IS (HECI MMIO 0x4[1]) and H_IE (HECI MMIO 0x4[0]) to 1 in order to receive an interrupt when TXE FW sets the CSE_RDY bit to 1. When H_RST transitions from 0 to 1, hardware clears the CSE_RDY and H_RDY bit to 0. The setting of H_RST is not needed nor desired after host partition reset. This will allow the TXE firmware to make the HECL interface ready without needing to wait for the IA FW to set H_RST. If the IA FW sets H_RST during a POST sequence where the TXE is also coming out of MOFF, the IA FW may experience a long (>100ms) delay prior to the TXE firmware being able to service the H_RST request.
4. Host reads the H_CSR once to ensure that the posted write to H_CSR in step 3 completes.
5. If IA FW did not set the H_IE to 1 in step 3, then IA FW now begins reading the CSE_CSR_HA MMIO register, polling for CSE_RDY_HRA bit (HECI MMIO 0xC[3]) to get set to 1
6. TXE perform HECL Interface Reset after receiving interrupt with reset request.
7. Since step 5 the host has either been polling CSE_RDY_HRA bit or waiting for a HECL host interrupt. Either of these two conditions will be satisfied now after TXE performed HECL Interface Reset and IA FW will proceed.
8. IA FW writes the H_CSR to clear H_RST to 0 and set H_RDY and H_IG to 1. IA FW is now ready to begin sending messages via the Host circular buffer.

28.3.4 Send/Receiving HECL Message

The TXE HECL interface must be initialized to prepare the host side of the TXE HECL interface to send and receive HECL messages as described in above sections.

The Circular Buffer Depth is allocated by TXE FW. The IA FW can find the TXE Circular Buffer Depth by checking H_CSR.H_CBD register (HECI MMIO 0x4[31:24]). The Circular Buffer Depth allows buffer depth values are of 2, 4, 8, 16, 32, 64, and 128.

The IA FW also needs to monitor the CSE_RDY (via CSE_CSR_HA.CSE_RDY_HRA in HECL MMIO 0xC [3]) value in 8 sec timeout loop. If CSE_RDY is set, it means the TXE firmware has successfully arranged the circular buffer in SRAM and circular buffer addresses and depth fields have all been initialized. At this occurrence, clear the host reset by setting H_CSR.H_RST = '0b' (HECI MMIO 0x4[4]) and prepare HECL ready by setting H_CSR.H_RDY (HECI MMIO 0x4[3]) and H_CSR.H_IG (HECI MMIO 0x4[2]) bits to '1b'. IA FW is now ready to begin sending messages via the Host Circular Buffer.

If the CSE_RDY is not set in time then this is an error case and IA FW should take TXE ERROR path.

ME Kernel Host Interface (MKHI) messages define the HECL messages for host and TXE Kernel communication. Refer Section [27.4](#). The HECL interface is designed around the concept of clients. Although IA FW is the producer of all aforementioned messages, the consumers within TXE space are different components with different client IDs.

28.3.4.1 Calculation of Filled Slots in a HECL Circular Buffer

The procedure for calculating the number of filled slots (entries) in a HECL circular buffer is the same for both the host circular buffer and TXE circular buffer. The



following steps allow calculation of the number of filled slots in a circular buffer, where "X" can be replaced with either "H" for host circular buffer or "CSE" for TXE circular buffer.

1. Cast ReadPointer, WritePointer as signed 8 bit quantity (-128 to +127).
2. Cast BufferDepth as an unsigned 8 bit quantity.
3. Cast FilledSlots as an unsigned 8 bit quantity.
4. Read X_CSR. (H_CSR: HECL MMIO 0x4; CSE_CSR: HECL MMIO 0xC)
5. Store X_CSR.X_CBRP (bit 15:8) in ReadPointer.
6. Store X_CSR.X_CBWP (bit 23:16) in WritePointer.
7. Store X_CSR.X_CBD (bit 31:24) in BufferDepth.
8. FilledSlots = (WritePointer – ReadPointer).

The following conditions may now be tested:

- If FilledSlots == 0, then the circular buffer is empty.
- If FilledSlots == BufferDepth, then the circular buffer is full.
- If FilledSlots > BufferDepth, then a circular buffer overflow occurred.

28.3.4.2 Calculation of Empty Slots in a HECL Circular Buffer

The procedure for calculating the number of empty slots (entries) in a HECL circular buffer is the same for both the host circular buffer and TXE circular buffer. The following steps allow calculation of the number of empty slots in a circular buffer, where "X" can be replaced with either "H" for host circular buffer or "CSE" for TXE circular buffer.

1. Cast EmptySlots as an unsigned 8 bit quantity.
2. Perform steps listed in Section [28.3.4.1](#).
3. EmptySlots = (BufferDepth – FilledSlots).

28.3.4.3 Sending Message to TXE

1. IA FW reads the H_CSR register (HECL MMIO 0x4) to determine the number of empty slots available in its circular buffer, as described above.
2. If there is not enough empty slots available in the host circular buffer, the host queues the request in an internal software queue and waits until there is room in the host circular buffer.
3. Once there is enough empty slots available in the host circular buffer, the host writes the message into the host circular buffer. To accomplish this, the host sequentially writes each DWord of the message to the H_CB_WW (Host Circular Buffer Write Window, HECL MMIO 0x0) MMIO register.
4. Host sets the H_IG bit (Host Interrupt Generate, HECL MMIO 0x4[2]) in the H_CSR register to 1, preserving the state of other bits, to generate an interrupt to the TXE.
5. Host reads the CSE_CSR_HA register (HECL MMIO 0xC) and checks that the CSE_RDY bit (HECL MMIO 0xC[3]) is set to 1. If the CSE_RDY bit is clear (0), a fatal error occurred during transmission of the message. The host must re-enter the initialization flow.
6. Upon the setting of the H_IG bit in step 4, hardware sets the CSE_IS and then clears the H_IG bit.
7. TXE FW is interrupted by an MSI (assuming HECL is configured to send the MSI to TXE FW).

The interrupt from TXE can be HECL message or an Interrupt with other reasons. IA FW interrupt service routine flow is described as below:



1. TXE FW prepares to send interrupt.
2. Hardware sets the H_IS (HECI MMIO 0x4[1]) and generates an interrupt to the host.
3. IA FW reads H_CSR register (HECI MMIO 0x4) for Interrupt reason
 - a. If H_D0I3C_IS (HECI MMIO 0x4[6]) bit is set, the interrupt was caused by TXE completion of a D0i3 command in progress. Clear H_DEVIDECLC_IS by writing a 1 to H_D0I3C_IS bit in the H_CSR.
4. If H_IS bit (HECI MMIO 0x4[1]) is set, interrupt was caused by TXE FW. IA FW clears the H_IS bit in the H_CSR by writing a 1 to the H_IS bit position in the H_CSR.
5. IA FW reads the CSE_CSR_HA
 - a. If CSE_RDY_HRA = 0 or CSE_RST_HRA = 1, TXE FW is requesting a HECI interface reset. IA FW should do initiate a HECI interface reset mentioned in Section [28.3.3](#) and exit interrupt service routine.
6. IA FW reads the CSE_CSR_HA (HECI MMIO 0xC) and determines the number of filled slots in the circular buffer available for reading (CSEFilledSlots), as described above.
7. If the TXE circular buffer is empty, the TXE has not sent a message. This interrupt is for another reason.
 - a. This interrupt indicates the TXE has completed reading the message(s) last transmitted by the host. The IA FW should be notified with the available slots in host HECI buffer for additional message transmission and exit interrupt routine.
8. If the TXE circular buffer has overflowed, the TXE has written too many slots in the circular buffer.
 - a. This is an error on the part of TXE FW. IA FW should do initiate a HECI interface reset mentioned in Section [28.3.3](#).
9. IA FW reads the CSE_CBRW (TXE Circular Buffer Read Window, HECI MMIO 0x8) register CSEFilledSlots times to get the current message data and moves it to a temporary buffer for dispatch to the appropriate IA FW component.
10. IA FW reads the CSE_CSR_HA. If the CSE_RDY bit (HECI MMIO 0xC[3]) is 0, then a TXE reset occurred during the transaction and the message should be discarded as bad data may have been retrieved from the CSE's circular buffer.
11. IA FW decodes the message header and signals the appropriate IA FW component that a message is ready for it to retrieve.
12. IA FW component sets the H_IG (Host Interrupt Generate, HECI MMIO 0x4[2]) bit in the H_CSR register to 1, preserving the state of other bits. This is done to signal the FW that the IA FW has consumed CSEFilledSlots from the host circular buffer.

28.3.5 HECI Link-Down Flow

The following flow is performed when the IA FW decide to put the HECI interface in a link-down state which allows Power Gating entry:

1. IA FW may send a HECI message to TXE indicating it would like to bring down the HECI link. In this case host waits for TXE FW to respond with an ack HECI message indicating it has accepted the request. At this point both IA FW and TXE FW will no longer send HECI messages.



2. IA FW writes to the H_CSR register, clearing the H_RDY bit (HECI MMIO 0x4[3]) and setting the H_IG bit to 1(HECI MMIO 0x4[2]).
 - a. IA FW may also set H_IS (HECI MMIO 0x4[1]) and H_IE (HECI MMIO 0x4[0]) to 1 in order to receive an interrupt when TXE FW resets the CSE_RDY (HECI MMIO 0xC[3]) bit to 0.
3. TXE FW is interrupted, prepares for Link-Down, and clear CSE_RDY bit.
4. IA FW can detect this by polling the CSE_RDY_HRA bit or interrupt.
 - a. If H_IS and H_IE are set in step 2a, IA FW is interrupted and runs the host ISR flow mentioned in above section to find out the CSE_RDY_HRA (HECI MMIO 0xC[3]) is cleared and the CSE_RST_HRA (HECI MMIO 0xC[4]) is also cleared so it understands that the TXE has done its part in bringing down the HECHI link.
5. IA FW completes the link-down flow by putting the HECHI interface into one of the followings states:
 - Put the HECHI device into D0i3: wait HECI1_D0I3C.H_D0I3C_CIP (HECI MMIO 0x800[0]) becomes 0, then set HECI1_D0I3C.H_D0I3C_IR (HECI MMIO 0x800[1]) and HECI1_D0I3C.H_D0I3C_I3 (HECI MMIO 0x800[2]). Then wait the D0i3 completion interrupt from TXE FW.
 - Put the HECHI device into D3: clear the BME (HECI PCI 0x4[2]), clear the MSI enable (HECI PCI 0x4[1]), set the PMCS.PME enable (HECI PCI 0x54[8]) if it is not HECHI disable, and set PMCS.PS to 2'b11(HECI PCI 0x54[1:0]).

28.3.6 Using HECHI2 in Windows* OS within SMM

When HECHI2 is used in Windows* OS with SMI, IA FW is expected to power manage HECHI2. From within SMM, IAFW must ensure HECHI2 is out of D0i3, use the HECHI2 and then put back the HECHI2 to D0i3.

The following flow is performed when the IA FW decides to take the HECHI interface out of link-down state, ex: HECHI2 is D0i3 in Windows* OS.

1. IA FW initiates the link-up flow by putting the HECHI interface into D0 states:
 - a. Move HECHI device out of D0i3: wait HECI1_D0I3C.H_D0I3C_CIP (HECI MMIO 0x800[0]) becomes 0, then set HECI1_D0I3C.H_D0I3C_IR (HECI MMIO 0x800[1]) and clear HECI1_D0I3C.H_D0I3C_I3 (HECI MMIO 0x800[2]). Then wait the D0i3 completion interrupt from TXE FW.
2. TXE FW performs related activities and generates an interrupt to host.
3. Host is interrupted and IA FW finds the TXE side of the HECHI interface ready for operation.
4. IA FW may send a HECHI message to TXE indicating it would like to bring up the HECHI link. IA FW waits for TXE FW to respond with an ack HECHI message indicating it has accepted the request.
5. At this point, IA FW can send necessary HECHI messages to TXE.
6. After completing all the HECHI messages, IA FW should put HECHI interface back to D0i3 as steps in previous section.

28.3.7 IA FW HECI Communication Guidance and Timeout

The IA FW should check the HFS register (Refer Section [28.2.3](#)) before sending out any HECI message. If HFS register shows any Current Working State except normal, IA FW should not generally send out any HECI message. I.e., when HFS register shows firmware is in “Recovery mode”, IA FW should avoid sending out any HECI message except for the DRAM Init Done message. There are some other exceptions specific to certain messages and firmware modes which are defined in the respective sections of this document.

All HECI messages require a response from CSE, IA FW will wait for a maximum of 5 seconds before next retry which includes re-sending of the message. IA FW will retry for a maximum of 3 times. If IA FW does not get any responses from TXE **after 3 retries with 50-sec timeout each**, then IA FW reads the HFS register to determine if TXE has run into the error state. If IA FW again detects a failure, then it should take TXE Error path.

28.3.8 Fixed Client Connection Protocol

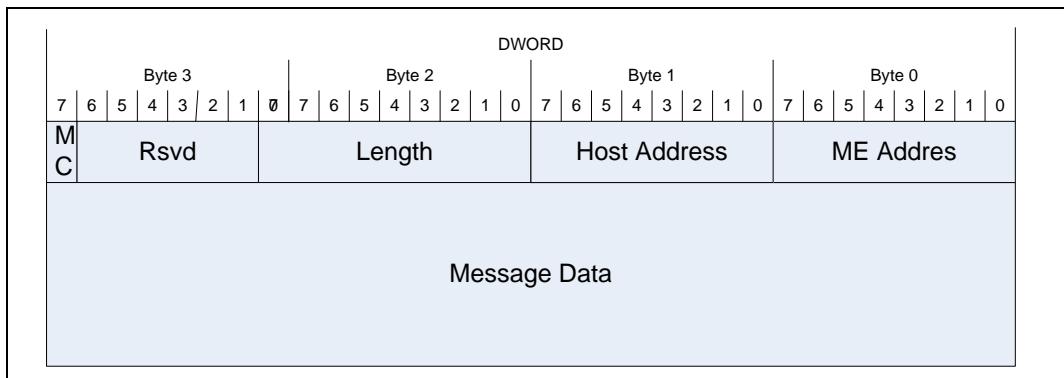
All IA FW interactions with TXE use HECI Fixed Client Connections, including the “MKHI – ME Kernel Host Interface”.

IA FW interacts with TXE over the TXE Interface. For details on HECI interface, message structure and programming guidelines, Refer “Intel® Management Engine - Host Embedded Controller Interface – Host Programming Specification (HPS)”.

This section gives an overview of the TXE fixed address HECI message structure as taken from HPS. If there are inconsistencies between this document and HECI-HPS then HECI-HPS will have the priority.

A general message format is used to pass data in the HECI circular buffers. The same message format is used for messages from the host to the TXE and from the TXE to the host. The message consists of a generic header and a data payload. The format and contents of the data payload are opaque to the host and firmware HECI drivers, and are only interpreted by the associated clients in the host and firmware domains.

The following figure shows the general message format of a message placed in an HECI circular buffer. The message consists of a message header (the first dword) and the message data. A message cannot be larger than the current circular buffer depth of the TXE interface. When a client sends HECI driver a message to transmit that is larger than the HECI circular buffer, the HECI driver will break this message into a set of smaller messages that can be transmitted one at a time through the HECI circular buffer and re-assembled by the receiving HECI driver. The last message that constitutes the last portion of a large message will have the MessageComplete (MC) bit in the message header is set.

Figure 28-2. Fixed Address HECI Header


The HECI message header is the 1st dword of any HECI message. It is described with the following C prototype:

Table 28-29. Definition for Fixed Address HECI Header

Field	Size (Bit)	Default	Description
ME Address	8	NA	<p>This is the logical address of the TXE client of the message. This address is assigned during TXE firmware initialization.</p> <ul style="list-style-type: none"> 0x00 HECI BUS message 0x01 Core message 0x02 Reserved 0x03 Reserved 0x04 WDT message 0x05 Reserved 0x07 MKHI message 0x08 Reserved 0x0B Hardware Asset message
HOST Address	8	NA	<p>This is the logical address of the Host client of the message. This address is assigned when the host client registers itself with the Host HECI driver.</p> <p>Address 0 is statically allocated for HECI BUS message protocol that assists in HECI BUS enumeration, power management, error and other bus control functionality.</p> <p>Address 1 is reserved.</p>

Field	Size (Bit)	Default	Description
Length	9	NA	<p>This is the message length in bytes, not including the HECL_MESSAGE_HEADER. A value of 0 indicates no additional bytes are part of the message.</p> <p>When a Client's message is fragmented into a multi-message HECL transfer, all HECL messages with MessageComplete field equal to 0 must have a Length field that is of dword granularity.</p> <p>Note that TXE interface transfers in unit of dwords. All extra padding bytes (at the end of the message) for this purpose will be discarded.</p> <p>Within a dword, bytes are in little endian order, i.e. byte 0 is bits 0 to 7, byte 1 is bits 8 to 15, and so on.</p>
Reserved	6	0x00	<p>These bits are reserved for future use. They must be zero but will be ignored by the HECL BUS driver to allow for future message backwards compatibility.</p>
Message Complete	1	0x00	<p>This bit is used to indicate that this is the last message of a multi message HECL transfer of a client's message that is larger than HECL circular buffer. If this bit is 0, at least one other message is expected to complete the client's message transfer. If this bit is 1, this message completes the transfer of a client's message. For client messages that can be transmitted in a single message, this bit will be set in that single message.</p> <p>Bit = 1: the message is either a complete message or the last fragment of a fragmented message.</p> <p>Bit = 0: the message is a fragment of a fragmented message, but is not that last fragment.</p>

28.4 BIOS CSE FW Interaction Overview

28.4.1 HECL Protocol

The following are important points about BIOS using HECL interface(s).

28.4.1.1 HECL1 PCI Device Configuration

The first HECL device is reserved for use by BIOS and the loaded OS. BIOS will use HECL1 interface to send messages to FW.



Step	Action	Register	Description
1	Lock HIDM	HECI1_HIDM	Set HIDM_L bit[2] = 1b to lock bits[1:0]
2	Enable PCI Device	HECI1 CFG Space	Set BAR register, Command register (BME,MSE) and initialize the PCI device.
3	Wait for CSE_RDY	HECI1_CSE_CSR_CSEA	CSE_RDY is expected to be set before host CPU is taken out of reset.
4	HECI Messaging	HECI1_CSE_CSR_CSEA, HECI1_H_CSR, HECI1_H_WW, HECI1_CSE_RW	Continue with HECI messaging.

28.4.1.2 HECI2-3

If HECI2 or HECI3 interface is used the host must perform an additional step of HECI link reset before using the HECI interface. This is due to BUP not setting CSE_RDY bit on HECI2/HECI3 device.

HECI2 and HECI3 are not normally used by the OS and therefore must be hidden from OS PCI bus driver enumeration.

28.4.1.2.1 HECI2 PCI Device Configuration

HECI2 will be used by BIOS for NVM offload in SMM. HECI2 must be setup for SMI and must be hidden from enumeration by OS PCI driver. BIOS will take the following steps to setup HECI2 for SMM use and hide from OS.

Step	Action	Register	Description
1	Enable HECI SMI	HECI2_HIDM	Set HIDM bits[1:0] = 10b for SMI.
2	Lock HIDM	HECI2_HIDM	Set HIDM_L bit[2] = 1b to lock bits[1:0]
3	Enable PCI Device	HECI2 CFG Space	Set BAR register, Command register (BME,MSE) and initialize the PCI device.
4	Hide PCI Config Space	PSF_3_AGNT_TO_SHDW_CFG_DIS_CE_RS0_D15_F2_OFFSET28	Set CfgDis bit[0] = 1
5	Link Reset HECI	HECI2_H_CSR/HECI2_CSE_CSR	Do link reset steps before using HECI2.
6	Enter D0I3	H_DEVIDLEC	Set the DEVIDLEC state to enable CSE to PG.

28.4.1.2.2 HECI3 PCI Device Configuration

HECI3 is hidden dependent if iTouch feature is available on the platform. BIOS need to do the following to enable/disable HECI3.

Step	Action	Register	Description
1	Lock HIDM	HECI2_HIDM	Set HIDM_L bit[2] = 1b to lock bits[1:0]. Note: By default register is set to MSI/Legacy interrupt type.
2	Setup PCI device	HECI CFG Space	Set BAR register, Command register (BME,MSE) and initialize the PCI device.
3	Get SKU rule to determine if touch is enabled.		Check for ship state in MBP. If missing send MKHI GET_RULE instead.
4	Set D0I3 state	HECI3_DEVIDLEC	Set DEVIDLE Bit[2] = 1. Note: This requires the PCI device to be initialized (ie BAR/MSE set).
5	Disable PCI Device	HECI3_CMD	Clear IOSE/MSE/BME BIT[0:2]
6	Put device in D3	HECI3_PMCS	Set PS Bit[1:0] = 3
7	Function Disable PCI Device	HECI3_SB_PCR_FUNC_DIS	Set FUNC_DIS in IOSFSB register.

28.4.1.3 HECI2 SMM Driver

BIOS should install a handler for HECI2 SMM operation that will handle reading and writing of NVM storage requests over HECI2. HECI2 SMM handler should follow HECI HPS with regard to interrupt handling and sending/receiving messages over CB.

Note: Since BIOS is using HECI2 only to send storage requests and after each request the CSE response is read from the CB. BIOS may omit sending PGI_ENTRY_REQ HBM to CSE before setting DEVIDLE bit in HECI2 DEVIDLEC register. This is a result of a concern on too many message overhead of HECI2 CB just to get storage requests. There is a desire to not stay in SMM more than needed and not generate extra SMIs.



28.4.1.4 HECI Error Handling

BIOS may encounter an error while communicating with CSE FW. The error could be the following:

1. CSE_RDY or H_RDY was cleared during message transmission to CSE or response from CSE.
2. A timeout of 5 seconds has elapsed waiting for CSE CB to clear (to send next message) or while waiting for response.
3. A timeout of 5 seconds has elapsed waiting for CSE_RDY to be set.
4. A CB overflow or underflow has occurred and is detected by Free/Filled slot calculation.

BIOS can choose to retry the communication by doing the following:

1. Follow the steps in HECI Link Reset Steps to re-initialize the link.
2. Wait for CSE_RDY to be set.
3. Retry the request to CSE.

BIOS may retry a total of 3 times before giving up. The 3 times number comes from the exception limit of CSE FW. After 3 exceptions have occurred CSE FW will enter recovery mode and HECI should not be tried.

28.4.1.5 Clearing MBP from CB

Since MBP has security sensitive entries the host will clear the MBP from the HECI CB. The reason for clearing out of HECI HW is due to HECI HW not guarding against over reading of the CB this means the host can replay the MBP if not overwritten by new HECI messages.

To clear the MBP the host does the following:

1. Sets the H_RST (BIT 4) and H_IG (BIT 2) in the HECI1_H_CSR.
2. Read back HECI1_H_CSR and verify H_RDY (BIT 3) is cleared.
3. On sending next HECI request read HECI1_CSE_CSR_HA and poll on CSE_RDY_HA bit (BIT 3) is set.
4. Clear H_RST (BIT 4), set H_RDY (BIT 3) and H_IG (BIT 2) in HECI1_H_CSR.
5. Send next message.

28.4.1.6 End of Post Error Handling

End of post is a message that indicates BIOS is no longer running and FW should lock down APIs and certain HW. Problems sending End Of Post can happen if CSE FW is in a hung state, either FW as a whole or process that handle End Of Post.

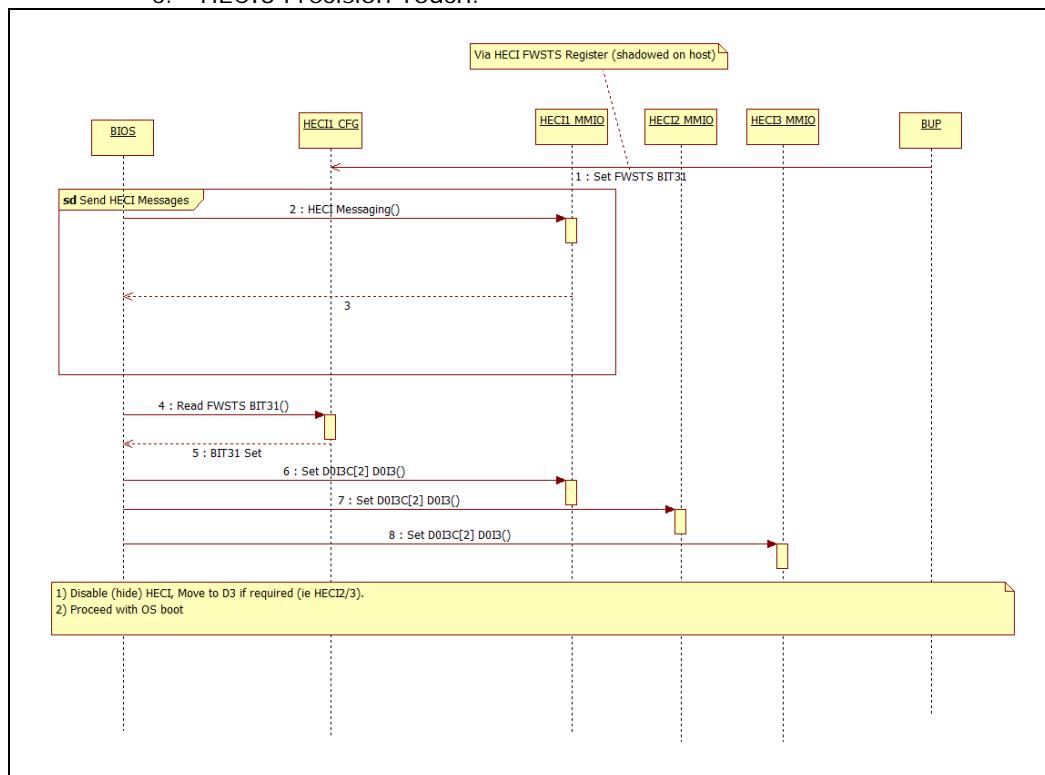
When BIOS encounters issues sending End Of Post BIOS should retry the command following the steps in HECI Error Handling. If those steps are not successful then BIOS must do the following.

1. Send the HECI_BUS_DISABLE command to the HECI FW driver.
2. Hide the HECI device using the HECI1_SB_PCI_FUNC_DIS register.
3. Continue boot.

28.4.1.7 DEVIDLE and Disable Requirement

BIOS will put all HECI devices into DEVIDLE before booting the OS. Below is a diagram showing for BIOS to check for D0I3 support and putting HECI devices into D0I3.

- Note:**
- 1) If BIOS disables HECI (via Private CR disable register) BIOS will first place HECI into DEVIDLE, then D3 state and then disable HECI.
 - 1) If BIOS does not disable HECI, BIOS must place device into D0I3 state before loading OS. It is suggested this be done at one of two options:
 - a. EOP-ACK received from CSE.
 - b. Boot prep services (if there is a need to use HECI before OS loads and after EOP-ACK).
 - 2) Any SW that uses HECI after BIOS places the device into DEVIDLE will be required to remove the device from DEVIDLE before use.
 - a. HECI OS SW
 - b. HECI2 BIOS NVM access.
 - c. HECI3 Precision Touch.



28.4.1.8 HECI MKHI Protocol

ME Kernel Host Interface is the main HECI client protocol to configure and interact with CSE. MKHI is registered at address 0x7 so that BIOS can send requests and receive responses without having a valid connection.

28.4.2 CSE Interaction Normal Boot Flow

The following interaction between BIOS/CSE happens during boot.



Stage	Operation	Interface	Description	Ref
IBBL/uCode	Secure boot load IBBL	RBP	Load IBBL into shared SRAM.	
IBBL	IBB Verification Done	RBP	Bits set to indicate IBB is verified. Jump to IBB and use HECL interface.	
IBB	Consume training data.		Training data copied to shared SRAM by RBP.	
IBB	Send DID to CSE	HECI1	BIOS sends DramInitDone to CSE FW CSE BUP loads CM0 main image.	
IBB	Get MBP	HECI1	BIOS requests MBP from CSE.	
IBB	Check for corruption entry.	HECI1	BIOS checks MBP for corruption entry and enters corruption boot path if present.	
IBB	Check for DNX OS bread crumb	HECI1	BIOS checks MBP for breadcrumb set requesting to enter DNX.	
IBB	Commit DRAM training data.	HECI1	Use NVM Write offload to commit training data to NVM.	
IBB	NVM handoff	HECI1	Request CSE to move NVM head to host.	
IBB	Load OBB		BIOS loads OBB	
IBB	SMM init done.	HECI2	Move to HECL2 for NVM access.	
OBB	SVN Update	HECI1	Check/commit SVNs for ARB.	
OBB	Lock Host SW Dir	HECI1	Host locks write to BIOS directory.	
OBB	EOP	HECI1	Host sends EOP to lock HECL APIs.	
OBB	EOS	HECI1	Before Boot Service Exit is called host sends EOS to lock down HECL1 NVM offload.	
OBB	HECL DOI3 Entry	HECL1/2/3	Before Boot Service Exit is called BIOS has placed HECL devices into DOI3 state.	
OS Load	SMI NVM Offload (cached)	HECI2	BIOS switches to HECL2 NVM offload. CSE is in cache mode (writes not committed).	

Stage	Operation	Interface	Description	Ref
OS	Storage Proxy	HECI1	Storage proxy is established by storage driver.	
OS	Commit writes	HECI1/HECI2	Storage writes are committed.	
OS	SMI NVM Offload	HECI1/HECI2	SMI NVM Offload is able to commit writes over HECI1 proxy.	
OS	OS unload	HECI1/HECI2	Storage proxy shuts down. Writes no longer committed to flash.	

28.4.3 Required HECI CSE Syncpoint

The following are the minimum syncpoints. They are required even in all CSE states.

SyncPoint	Description	Reason
DRAM INIT DONE	Indication to CSE that UMA is available.	Signal to transition CSE to UMA. CSE will transfer its BUP code pages and page backing to UMA in states where CSE update is permitted.
CORE BIOS DONE	Lock down HECI1 Write Data API.	Locks down HECI1 NVM Write offload.
DATA CLEAR LOCK	Locks down CSE initiated data clear.	Locks down the CSE data clear API from later stages of BIOS.
END OF POST	Locks down various APIs from OS use.	Locks out HECI1 NVM and other sensitive CSE APIs.
END OF SERVICE	On EMMC based systems EOS moves the NVM head to the host.	SPI CSE boot platforms – optional as SPI HW is not single head. EMMC CSE boot platforms required to move NVM to host and lock at host.

28.4.4 CSE Interaction DNX

The following cases affect DNX and cause BIOS to send HECI messages to FW.



28.4.4.1 CSE Corruption

In this case CSE detects corruption and does not load main image.

Stage	Operation	Interface	Description	Ref
CSE	CSE detects corruption.		CSE detects corruption and initiates global reset.	
CSE	Transfer NVM to host.		CSE BUP enters corruption case, does not load main image and transfers NVM to host before sending MBP.	
IBB	Get MBP and consume corruption entry.	HECI1	IBB gets the MBP from CSE and a corruption entry is present. BIOS takes CSE corruption path.	
IBB	Enter DNX	HECI1	IBB uses SET_FILE to set RBE breadcrumb to enter DNX.	
IBB	Global Reset		IBB initiates global reset to enter DNX.	

28.4.4.2 BIOS Detected Corruption

BIOS detects corruption in OBB or OS breadcrumb is set.

Stage	Operation	Interface	Description	Ref
IBB	CSE detects corruption.		CSE detects corruption and initiates global reset.	
CSE	Transfer NVM to host.		CSE BUP enters corruption case, does not load main image and transfers NVM to host before sending MBP.	
IBB	Get MBP and consume DNX breadcrumb. ¹	HECI1	IBB gets MBP and checks if the DNX breadcrumb is present/set to request DNX entry.	
IBB	IBB detects corruption in OBB ¹		IBB detects OBB corruption and wants to enter DNX.	
IBB	Enter DNX	HECI1	IBB uses SET_FILE to set RBE breadcrumb to enter DNX.	
IBB	Global Reset		IBB initiates global reset to enter DNX.	

NOTE: It is possible either MBP is present or OBB corruption is detected or both may be present. The end result is DNX is entered as a result of IBB sending MKHI message to set RBE DNX breadcrumb.

28.4.5 Other Configuration Messages

The BIOS is expected to receive the majority of its configuration data through MBP however on first boot after flash MBP may not be fully populated or for special BIOS flows extra MKHI requests must be sent (since MBP only contains data needed for most boots) or to set configuration parameters for various CSE features

These messages are detailed in this doc and are sent at the discussion of the BIOS designer and requirement.

28.4.6 NVM Access and Handoff

The NVM storage head is only accessible to one platform entity at time, either host or CSE FW. The NVM head will be programmed to be accessible by CSE FW on boot so that CSE FW can load its boot image and enable the boot of the platform. CSE will transition the NVM storage head to the host during boot where it will remain.

For storage proxy commands before end of services message the CSE will reprogram the head to be CSE accessible and restore the head on response FFW and request DNX.

28.4.7 Recovery Available MKHI API

If CSE FW NFTP is corrupt the FTP will load up to BUP process. Since CMO CSE FW is not available only BUP handled MKHI API are available. If host sends an unhandled MKHI request BUP will NACK with error. APIs needed to boot the platform and perform a recovery are required to be supported in CSE BUP.

The list of available APIs are:

- RBP
- Request Device Ownership
- HECl1 Read Data
- Lock Host SW Directory Writes
- Dram Init Done
- ME BIOS Payload
- End Of Post
- End Of Services
- IFWI Prepare For Update
- Data Clear
- Get Current Boot Media

Each MKHI definition in this document has an entry that explains what process it belongs to. If it belongs to BUP it will be available. Refer command details for explanation of limitation in recovery if any.



28.4.8 Power Management

28.4.8.1 G3, S5, S4 Exit and Entry to S0/Reset Exit and Entry to S0

The BIOS is expected to interact with CSE for these PM flows and do the following:

- 1) Secure Boot flow if applicable.
- 2) Initialize UMA/IMRs and send Dram Init Done to CSE.
- 3) Read ME BIOS Payload and perform any boot initialization of CSE.
- 4) Any other required HECI related interaction that may be required.
- 5) Send End Of Post.
- 6) Put HECI devices into D0I3.
- 7) Disable any HECI devices that cannot be enumerated by the OS.

28.4.8.2 S3 Exit Transition to S0

- 1) Initialize UMA/IMRs and send Dram Init Done to CSE.
- 2) Read ME BIOS Payload.
- 3) Put HECI device into D0I3.
- 4) Disable any HECI devices that cannot be enumerated by the OS.

28.5 FW Boot Mode

The following section describes how to determine the FW boot mode and which FWSTS register to read. It is recommended to first read the current state and then follow the action column to proceed.

28.5.1 Determine Current State

The following defines the current state value read from HECI1_CSE_FS_CURSTATE field. The value will indicate the boot mode of CSE. BIOS should check the CURSTATE field after receiving DRAM INIT DONE ACK from CSE.

Value	Name	Mode	Action
0	RESET	CSE is coming out a reset.	Wait for CSE to exit the reset.
1	INIT	CSE is initializing BUP.	Wait for CSE to finish initializing.
2	Recovery	CSE has entered recovery state.	Check NFTP_FAILURE bit in FWSTS2. See recovery section below.
5	Normal	CSE BUP has booted successfully.	Check OPMODE register to determine BOOT mode. Refer normal section below.
14	Halt	CSE BUP has halted due to an error.	Check ERRCODE register for more debug information. Refer halt section below.



28.5.2 Determine Operation Mode

The following defines the operation mode value read from HECI1_CSE_FS_OPMODE field. The mode will indicate the mode the CSE is operating in. BIOS should check the OPMODE field after reading NORMAL CURSTATE field.

Value	Name	Mode	Action
0	HECI1_CSE_FS_OPMODE_NORMAL	CSE is operating in a normal state.	BIOS can use enabled CSE FW features.
4	HECI1_CSE_FS_OPMODE_SECURITY_OVERRIDE_JUMPER	CSE is halted in BUP ready for update of ifwi.	BIOS or host tool can proceed to flash the NVM. Remove FDO pin strap jumper and reboot to exit mode.
6	HECI1_CSE_FS_OPMODE_IFWI_UPDATE_MODE	CSE has restarted and halted in BUP.	BIOS or host tool can proceed to flash NVM after BUP responds with ready to IFWI PREPARE FOR UPDATE Message. Global reset to exit mode. Note: HECI1_CSE_FS_FW UPDATERESTATE will indicate if firmware update has started. This is requested for debug.

28.5.3 Recovery Mode

This boot mode CSE has entered an error state and is disabled. BIOS should check the HECI1_CSE_GS1_NFTP_FAILURE bit in FWSTS2 to determine how to proceed.

In the case the bit is not set and a power cycle does not result in changing the state then the platform will have to be refurbished. The prompt below may be displayed by BIOS to have user try a power cycle to resolve.

Recommended User Prompt for value 0:

A power cycle may resolve the issue.

If the issue persists then contact the manufacturer for diagnosis.

Value	Name	Mode	Action
0	HECI1_CSE_GS1_NFTP_LOAD_SUCCESS	The CSE has booted into an error state due not due to failed firmware update.	BIOS should prompt the user.



Value	Name	Mode	Action
1	HECI1_CSE_GS1_NFTP_LOAD_FAILURE	The CSE has booted into an error state after detecting NFTP is corrupt. This is likely due to a failed firmware update.	BIOS should proceed to retry the firmware update process.

28.5.4 Error Mode

The following defines the error values read from HECI1_CSE_FS_ERRCODE field.

In the case the CSE boots into the error state and a power cycle does not result in changing the state then the platform will have to be refurbished. The prompt below may be displayed by BIOS to have user try a power cycle to resolve.

Recommended User Prompt:

A power cycle may resolve the issue.

If the issue persists then contact the manufacturer for diagnosis.

Value	Name	Mode	Action
0	HECI1_CSE_FS_ERRCO DE_NOERROR	The CSE is not in an error state.	BIOS can use enabled CSE FW features.
4	HECI1_CSE_FS_ERRCO DE_DEBUG_ERROR	The CSE is halted in BUP. NOTE: This will be set when CSE has booted to FDO or ifwi update mode in order to allow host to update ifwi.	Check HECI1_CSE_FS_OPMODE and follow section Determine Operation Mode.
1, 2, 3, 5	See HECI1_CSE_FS_ERRCODE	CSE is in an error state.	The values here are used to help in debug.

28.6 Firmware Status Definitions (FWSTS DEFS)

28.6.1 HECI 1 FWSTS1 DEFS

BIT	Field Name	Field Values	Description
0-3	HECI1_CSE_FS_CURSTATE	0-0xE	Current Working State
4	HECI1_CSE_FS_MFGMODE	0-0x1	Current manufacturing mode
5	HECI1_CSE_FS_FPTSTATE	0-0x1	Indicates status of flash partition table.

BIT	Field Name	Field Values	Description
6-8	HEC1_CSE_FS_OPSTATE	0-0x7	Current Operational State of CSE FW
9	HEC1_CSE_FS_INITSTATE	0-0x1	Indicates if CSE FW is fully initialized.
10	HEC1_CSE_FS_BUPLOADSTATE	0-0x1	Indicates if BUP loaded successfully.
11	HEC1_CSE_FS_FWUPDATESTATE	0-0x1	Indicates if CSE FWU is in progress.
12-15	HEC1_CSE_FS_ERRCODE	0-0x5	Indicates if CSE encountered an error and has stopped.
16-19	HEC1_CSE_FS_OPMODE	0-0x5	The current mode of operation.
20-23	HEC1_CSE_FS_RSTCOUNT	0-0x3	CSE error reset counter.
24	HEC1_CSE_FS_BOOT_OPT	0-0x1	CSE Boot options available.
25	HEC1_CSE_FS_BIST_FINISHED	0-0x1	Indicates if BIST is finished.
26	HEC1_CSE_FS_BIST_TEST_STATE		Current BIST test state.
27	HEC1_CSE_FS_BIST_HOST_PD	0-0x1	BIST reset request
28-29	HEC1_CSE_FS_CURR_PWR_SRC	0-0x2	Current Power Source
30	HEC1_CSE_FS_D3	0	HEC1 FW RTD3 Support Enabled
31	HEC1_CSE_FS_DEVIDLE	0x1	HEC1 FW DEVIDLE Support Enabled

28.6.1.1 HEC1_CSE_FS_CURSTATE

Describes the current working state of the CSE FW.

This is what the FW is doing at the moment. Further info about the progress can be found in the extended (EXT) state data. This field describes the current working state of the Intel(R) CSE firmware. In spoken language, the "Current Working State" indicates what firmware is doing at this moment. Further information about the progress within the Current Working State can be found in the Extended Status Data field of this register.

Value	Name	Description
0	HEC1_CSE_FS_CURSTATE_RESET	CSE FW is in a reset state.
1	HEC1_CSE_FS_CURSTATE_INIT	FW is starting and beginning its load
2	HEC1_CSE_FS_CURSTATE_RECOVERY	FW is recovering from an image update failure
3	HEC1_CSE_FS_CURSTATE_TEST	FW is executing in an HVM test mode



Value	Name	Description
4	HEC1_CSE_FS_CURSTATE_FW_DISABLED	Legacy. Use OpMode Temp Disable
5	HEC1_CSE_FS_CURSTATE_NORMAL	CSE in a normal working state
6	HEC1_CSE_FS_CURSTATE_DISABLE_WAIT	CSE disabled and waiting to timeout before disabling host
7	HEC1_CSE_FS_CURSTATE_OP_STATE_TRANS	CSE is performing an operational state transition
8	HEC1_CSE_FS_CURSTATE_INVALID_CPU_PLUGGED_IN	identifying that a disallowed CPU SKU is populated in the platform as determined by PCH HW SKU
9	HEC1_CSE_FS_CURSTATE_MISMATCH_IN_PCH_DID	identifies if user is trying to use wrong PCH SKU Emulation via FITC vs what's the actual HW Type
10	HEC1_CSE_FS_CURSTATE_SECURE_BOOT_ACN_FAILURE	Secure Boot failure, ACM module failure
11	HEC1_CSE_FS_CURSTATE_SECURE_BOOT_SM_FAILURE	Secure Boot failure, SM module failure
12	HEC1_CSE_FS_CURSTATE_SECURE_BOOT_SM_TIMER_ZERO	Secure Boot failure, SM Timer zeroed out, infinite wait
13	HEC1_CSE_FS_CURSTATE_SB_ENABLED_PCH_PLUGGED_IN	Secure Boot Enabled, PCH is plugged in
14	HEC1_CSE_FS_CURSTATE_HALT	CSE detects fatal error

28.6.1.2 HECI1_CSE_FS_MFGMODE

It describes if the platform is in its manufacturing mode.

This bit is set when GVLD is 0 or the chipset descriptors are not locked. This bit can be used by host to inform/warn that the user has not readied the system for production yet.

Value	Name	Description
0	HEC1_CSE_FS_MFGMODE_DISABLED	FW is not in manufacturing mode and is ready for production
1	HEC1_CSE_FS_MFGMODE_ENABLED	FW is in manufacturing mode where it can easily be configured

28.6.1.3 HECI1_CSE_FS_FPTSTATE

FPTSTATE describes the state of the Flash Partition Table in the image.

This bit is set when the FW has detected that the Flash Partition Table (FPT) is not valid. In this case the FW will try to brute force locate a valid FW image, but may have issue with any configuration data. Primarily seen in a min-sku environment.

Value	Name	Description
0	HECI1_CSE_FS_FPTSTATE_VALID	Flash Partition was found in a valid state
1	HECI1_CSE_FS_FPTSTATE_INVALID	Flash Partition was found in an invalid state

28.6.1.4 HECI1_CSE_FS_OPSTATE

OPSTATE describes current operational state of the FW. This is what state the FW is in at that moment. Further info about the progress can be found in the extended (EXT) state data.

Value	Name	Description
0	HECI1_CSE_FS_OPSTATE_PREBOOT	Operating from ROM prior to determining op state
1	HECI1_CSE_FS_OPSTATE_CMO_UMA	Operating in CMO with UMA memory
2	HECI1_CSE_FS_OPSTATE_CMO_PG	Operating in CMO_PG with UMA memory
3	Reserved	Reserved
4	HECI1_CSE_FS_OPSTATE_CM3_NOUMA	Operating in CM3 with No UMA memory
5	HECI1_CSE_FS_OPSTATE_CMO_NOUMA	Operating in CMO with No UMA memory
6	HECI1_CSE_FS_OPSTATE_BUP	Operating from BUP prior to determining op state
7	HECI1_CSE_FS_OPSTATE_CMO_NOUMA_ERR	Operating in M0 with No UMA and a FW error occurred

28.6.1.5 HECI1_CSE_FS_INITSTATE

INITSTATE describes current initialization state of the FW. When FW Init is completed this flag is set, otherwise FW is in the process of initializing. Further info about the progress can be found in the extended (EXT) state data.

Value	Name	Description
0	HECI1_CSE_FS_INITSTATE_INITIALIZING	CSE FW is initializing
1	HECI1_CSE_FS_INITSTATE_COMPLETED	CSE FW has completed boot.

28.6.1.6 HECI1_CSE_FS_BUPLOADSTATE

BUPLOADSTATE describes any load issues found when initially loading the BUP module. This indicates that no valid image could be found in the Fault Tolerant Partition. This can occur due to fault during FW update or possible image corruption.



In FW update the code should be backed up, so a failure like this cause the FW to looks for a backup image to load.

Value	Name	Description
0	HECI1_CSE_FS_BUPLOADSTATE_SUCC ESS	BUP loaded successfully from FT partition
1	HECI1_CSE_FS_BUPLOADSTATE_FAILU RE	BUP module could not be loaded from the FT partition

28.6.1.7 HECI1_CSE_FS_FWUPDATESTATE

FWUPDATESTATE describes the state of FWUPDATE. This indicates if the FW update is in progress or not.

Value	Name	Description
0	HECI1_CSE_FS_FWUPDATE_STATE_IDLE	FW update is in idle state
1	HECI1_CSE_FS_FWUPDATE_STATE_IN_PROGRES S	FW update is in progress

28.6.1.8 HECI1_CSE_FS_ERRCODE

ERRCODE describes a fatal error state if the FW encounters. Set to a non-zero value if the FW encounters a fatal error and has stopped normal operation. Further information about the location of the failure can be found in the current state and extended state data.

Value	Name	Description
0	HECI1_CSE_FS_ERRCODE_NOERROR	FW is operation in current state
1	HECI1_CSE_FS_ERRCODE_UNKNOWN_FAIL URE	FW has experienced an uncategorized error
2	HECI1_CSE_FS_ERRCODE_DISABLED	FW forced halted for non production use
3	HECI1_CSE_FS_ERRCODE_IMAGE_FAILURE	CSE flash image and recovery image is invalid
4	HECI1_CSE_FS_ERRCODE_DEBUG_ERROR	Error occurred and extended status should be used (note: Defined same as FATAL_ERROR)
4	HECI1_CSE_FS_ERRCODE_FATAL_ERROR_I DENTIFIED	FW detected error state and extended status can be referred (note: Defined same as DEBUG_ERROR).
5	HECI1_CSE_FS_ERRCODE_PG_EXIT_IMAGE _FAILURE	Error occurred during PG exit flow

28.6.1.9 HECI1_CSE_FS_OPMODE

OPMODE describes the mode of operation. The mode of operation defines which configured mode the FW is working.

Value	Name	Description
0	HECI1_CSE_FS_OPMODE_NORMAL	Normal operating mode
1	HECI1_CSE_FS_OPMODE_MINSKU	Min-SKU operating mode
2	HECI1_CSE_FS_OPMODE_ALTDISABLE	Alt Disable operating mode
3	HECI1_CSE_FS_OPMODE_TEMP_DISABLE	Temporary/Soft Disable operating mode
4	HECI1_CSE_FS_OPMODE_SECURITY_OVER_RIDE_JUMPER	FW set into unsecured op mode by H/W jumper
5	HECI1_CSE_FS_OPMODE_SECURITY_OVER_RIDE_HECI_MSG	FW set into unsecured op mode by HECI msg

28.6.1.10 HECI1_CSE_FS_RSTCOUNT

RSTCOUNT describes the number of CSE reset.

Value	Name	Description
0	HECI1_CSE_FS_CSERSTCNT_0	0 CSE Resets
1	HECI1_CSE_FS_CSERSTCNT_1	1 CSE Resets
2	HECI1_CSE_FS_CSERSTCNT_2	2 CSE Resets
3	HECI1_CSE_FS_CSERSTCNT_3	3 CSE Resets

28.6.1.11 HECI1_CSE_FS_BOOT_OPT

BOOT_OPT describes if the boot option is available.

Value	Name	Description
0	HECI1_CSE_FS_BOOT_OPT_INVALID	BOOT Option is unavailable.
1	HECI1_CSE_FS_BOOT_OPT_AVAILABLE	BOOT Option is valid

28.6.1.12 HECI1_CSE_FS_BIST_FINISHED

BIST_TEST_STATE describes the BIST test status bist_mgr sets to 0 when BIST is starting and sets to 1 when BIST has finished. MEManuf tool can then call GetResults HECI command to get BIST results.

Value	Name	Description
0	HECI1_CSE_FS_BIST_TEST_STATE_STARTING	BIST TEST is starting
1	HECI1_CSE_FS_BIST_TEST_STATE_FINISHED	BIST TEST is finished



28.6.1.12.1 HECI1_CSE_FS_BIST_TEST_STATE

Meaningful for usage model 1 only.

Value	Name	Description
0	HECI1_CSE_FS_BIST_RESET_REQUEST_NOT_READY	One or more tests have failed.
1	HECI1_CSE_FS_BIST_RESET_REQUEST_READY	All category 1 tests passed.

28.6.1.13 HECI1_CSE_FS_BIST_HOST_PD

BIST sets to indicate host SW should power down to trigger CM0/S0 -> CM3/Sx in full BIST.

Value	Name	Description
0	HECI1_CSE_FS_BIST_HOST_PD_NOT_REQUESTED	No request to power down
1	HECI1_CSE_FS_BIST_HOST_PD_REQUESTED	Request to power down.

28.6.1.14 HECI1_CSE_FS_CURR_PWR_SRC

PMDrv will set this field the current power source.

Value	Name	Description
0	HECI1_CSE_FS_CURR_PWR_SRC_UNKNOWN	Unknown power source
1	HECI1_CSE_FS_CURR_PWR_SRC_AC	AC
2	HECI1_CSE_FS_CURR_PWR_SRC_DC	DC

28.6.1.15 HECI1_CSE_FS_D3

Indicates if HECI1 FW driver supports RTD3 for host SW.

Value	Name	Description
0	HECI1_CSE_FS_D3_NOT_SUPPORTED	RTD3 not supported.

28.6.1.16 HECI1_CSE_FS_DEVIDLE

Indicates if HECI1 FW driver supports DEVIDLE for host SW.

Value	Name	Description
0	HECI1_CSE_FS_D3_NOT_SUPPORTED	DEVIDLE not supported. Note only on legacy platforms.
1	HECI1_CSE_FS_D3_SUPPORTED	DEVIDLE Supported.

28.6.2 FWSTS2 DEFS

BIT	Field Name	Field Values	Description
0	HECI1_CSE_GS1_NFTP_FAILURE	0-0x1	NFTP failure detected.
1-2	HECI1_CSE_GS1_ICC	0-0x3	ICC programming done by CSE FW.
3	HECI1_CSE_GS1_INVOKE_MEBX	0-0x1	CSE requested MEBX.
4	HECI1_CSE_GS1_CPU_REPL	0-0x1	CPU replacement warning.
5	Reserved	0	Reserved
6	HECI1_CSE_GS1_MFS_FAILURE	0-0x1	MFS failure
7	HECI1_CSE_GS1_DF_WRST_REQ	0-0x1	CPU replacement warm reset request.
8	HECI1_CSE_GS1_CPU_REPL_VALID	0-0x1	CPU replacement valid
9	HECI1_CSE_GS1_LP_STATE	0-0x1	CSE Low Power state.
10	HECI1_CSE_GS1_PG_STATE	0-0x1	CSE Power Gate State. (Deprecated).
11	HECI1_CSE_GS1_IPU_NEEDED	0-0x1	CSE IPU State.
12	HECI1_CSE_GS1_FWU_FORCED_SAFE_BOOT	0-0x1	FWU Safe Boot status
13	Reserved	0	Reserved
14	HECI1_CSE_FS_BIST_HOST_PD	0-0x1	IFU fault tolerant state.
15	HECI1_CSE_GS1_LISTEN_CHANGE	0-0x1	Indicate HECI FW listener status.
16-23	HECI1_CSE_GS1_STATUS	0-0xFF	Current status.
24-27	HECI1_CSE_GS1_PM	0-0xF	Last PM status.
28-31	HECI1_CSE_GS1_PHASE	0-0x8	Current phase.

28.6.2.1 HECI1_CSE_GS1_NFTP_FAILURE

This bit is set by FW when any part of the non fault tolerant flash partition is found to be invalid and the FW enters into a current working state of recovery mode. A FW update can recover the FW to a normal working state ONLY IF the FW current working state is Recovery and the FW is not in an error state.



Value	Name	Description
0	HECI1_CSE_GS1_NFTP_LOAD_SUCCESS	Non Fault Tolerant Flash Partition is valid
1	HECI1_CSE_GS1_NFTP_LOAD_FAILURE	Non Fault Tolerant Flash Partition is found to be invalid

28.6.2.2 HECI1_CSE_GS1_ICC

ICC describes the state of ICC programming done by FW more significant bit describes if there is valid data on SPI least significant bit describes if registers were populated with new values

Value	Name	Description
0	HECI1_CSE_GS1_ICC_SPI_OEM_DATA_ERROR	No valid data read from SPI
1	HECI1_CSE_GS1_ICC_SPI_OEM_DATA_OK	Valid data read from SPI
2	HECI1_CSE_GS1_ICC_OEM_DATA_PROG_ERROR	no ICC programing
3	HECI1_CSE_GS1_ICC_OEM_DATA_PROG_OK	ICC programming OK

28.6.2.3 HECI1_CSE_GS1_INVOKE_MEBX

INVOKE_MEBX describes the state of MEBx invocation requirement programming done by BUP. The '1b' means the BIOS has to launch CoreMEBXEntry() which is a service provided by IntelR MEBX protocol.

Value	Name	Description
0	HECI1_CSE_GS1_INVOKE_MEBX_DISABLED	No MEBx invocation request
1	HECI1_CSE_GS1_INVOKE_MEBX_ENABLED	requested to launch CoreMEBXEntry().

28.6.2.4 HECI1_CSE_GS1_CPU_REPL

CPU_REPL describes the state of CPU replace programming done by BUP.

Value	Name	Description
0	HECI1_CSE_GS1_CPU_REPL_NO_DETECTION	CPU replacement is not detected.
1	HECI1_CSE_GS1_CPU_REPL_DETECTED	CPU replacement detected.

28.6.2.5 HECI1_CSE_GS1_MFS_FAILURE

MFS_FAILURE describes the state of MFS failure programming done by FW

Value	Name	Description
0	HECI1_CSE_GS1_MFS_NO_FAILURE	No MFS failure detected.
1	HECI1_CSE_GS1_MFS_FAILURE_DETECTED	MFS failure detected.

28.6.2.6 HECI1_CSE_GS1_DF_WRST_REQ

DF_WRST_REQ describes the state of warm reset request programming done by FW..

Value	Name	Description
0	HECI1_CSE_GS1_DF_WRST_NO_REQUEST	No warm reset request.
1	HECI1_CSE_GS1_DF_WRST_REQUESTED	Warm reset requested.

28.6.2.7 HECI1_CSE_GS1_CPU_REPL_VALID

CPU_REPL_VALID describes the valid state of CPU replace programming done by FW

Value	Name	Description
0	HECI1_CSE_GS1_CPU_REPL_INFO_INVALID	CPU replacement bit is invalid.
1	HECI1_CSE_GS1_CPU_REPL_INFO_VALID	CPU replacement bit is valid.

28.6.2.8 HECI1_CSE_GS1_LP_STATE

LP_STATE describes the state of Low Power State programming done by FW

Value	Name	Description
0	HECI1_CSE_GS1_LP_STATE_DISABLED	Low power state is disabled.
1	HECI1_CSE_GS1_LP_STATE_ENABLED	Low power state is enabled.

28.6.2.9 HECI1_CSE_GS1_PG_STATE

PG_STATE describes the state of power gating programming done by FW

Value	Name	Description
0	HECI1_CSE_GS1_PG_STATE_DISABLED	PG state is disabled
1	HECI1_CSE_GS1_PG_STATE_ENABLED	PG state is enabled.



28.6.2.10 HECI1_CSE_GS1_IPU_NEEDED

GS1_IPU_NEEDED describes the state of IPU needed programming done by FW

Value	Name	Description
0	HECI1_CSE_GS1_IPU_NEEDED_INVALID	IPU needed is invalid.
1	HECI1_CSE_GS1_IPU_NEEDED_VALID	IPU needed is valid.

28.6.2.11 HECI1_CSE_GS1_FWU_FORCED_SAFE_BOOT

FWU_FORCED_SAFE_BOOT describes the state of FWupdate forced safe boot programming done by FW.

Value	Name	Description
0	HECI1_CSE_GS1_FWU_FORCED_SAFE_BOOT_DISABLED	Normal boot
1	HECI1_CSE_GS1_FWU_FORCED_SAFE_BOOT_ENABLED	FWU force the system to boot in safe boot.

28.6.2.12 HECI1_CSE_FS_BIST_HOST_PD

28.6.2.13 HECI1_CSE_GS1_LISTEN_CHANGE

Registration change for CSE FW event listener. A process has registered or unregistered from getting events.

Value	Name	Description
0	HECI1_CSE_GS1_LISTEN_NO_CHANGE	No change in registration.
1	HECI1_CSE_GS1_LISTEN_CHANGE	Change in registration.

28.6.2.14 HECI1_CSE_GS1_STATUS

Current status against the phase of operation. See HECI1_CSE_GS1_PHASE values.

28.6.2.14.1 HECI1_CSE_GS1_PHASE_ROM

Value	Name	Description
0x00	ROM_BEGIN	Covers a number of steps early in ROM flow before entering main
0x01	ROM_INIT_HW	Initialize memory map
0x02	ROM_INIT TPM EST BIT	Initialize the TPM Establishment bit
0x03	ROM_INIT_SUSRAM	Initialize SusRAM

Value	Name	Description
0x04	ROM_GET_FUSES	Get Fuse values
0x05	ROM_DERIVE_UMCHID	Derive chipset keys
0x06	ROM_DISABLE	Fuses/Straps indicate ME disabled
0x07	ROM_INIT_HECI	Initialize HECI buffers
0x08	ROM_FIND_IMAGE	Find a FPT/Manifest image
0x09	ROM_MANIFEST_FOUND	Validate manifest found
0x05	ROM_DERIVE_CHIPKEYS	Derive chipset keys
0x0A	ROM_LOAD_MODULE	Load module found in manifest
0x0B	ROM_CALL_NEXT_MODULE	Transfer control over to new module
0x0C	ROM_FIND_FPT	Locate the FPT within memory
0x0D	ROM_FIND_CODE_PARTITION	Locate the Code partition within the FPT
0x0E	ROM_INIT_MEMORY	Initialize Paging logic
0x0F	ROM_PATCH_FAILURE	Function designed to be patched was not prior to use.
0x10	ROM_PAGE_FAULT	H/W Page fault occurred
0x11	ROM_FPT_INVALID	FPT invalid
0x12	ROM_BRUTE_FORCE_SCAN	Brute force scan ensued
0x13	ROM_IDLM_FOUND	IDLM module found at the end of NFTP
0x14	ROM_PG_EXIT_START	ROM PG exit flow start
0x15	ROM_PG_EXIT_LOAD_KERNEL	ROM PG exit uKernel load

28.6.2.14.2HECI1_CSE_GS1_PHASE_RBE

Not used.

28.6.2.14.3HECI1_CSE_GS1_PHASE_UKERNEL

Value	Name	Description
0x00	KERNEL_BEGIN	Entry into KERNEL Module
0x01	KERNEL_APPDEFINE_START	Entry into ThreadX Application Define
0x02	KERNEL_APPDEFINE_END	Exit ThreadX Application Define



Value	Name	Description
0x03	KERNEL_PM_GO_S3_RCVD	Received S3 entry MSI from PMC
0x04	KERNEL_PM_GO_S4_RCVD	Received S4 entry MSI from PMC
0x05	KERNEL_PM_GO_S5_RCVD	Received S5 entry MSI from PMC
0x06	KERNEL_PM_POWER_DOWN_RCVD	Received UPD entry MSI from PMC
0x07	KERNEL_PM_PWR_CYCLE_RST_RCVD	Received PCR entry MSI from PMC
0x08	KERNEL_PM_NON_PWR_CYCLE_RST_R_CVD	Received NPCR entry MSI from PMC
0x09	KERNEL_PM_HOST_WAKE_RCVD	Received host wake MSI from PMC
0x0A	KERNEL_PM_PWR_SRC_AC	power source AC
0x0B	KERNEL_PM_PWR_SRC_DC	power source DC
0x0C	KERNEL_PM_DID_RCVD	Received DRAM Init Done
0x0D	KERNEL_UNKNOWN_FLASH_DEVICE	VSCC Data not found for flash device
0x0E	KERNEL_INVALID_VSCC_TABLE	VSCC Table is not valid
0x0F	KERNEL_INVALID_FLASH_PARTITION_BOUNDARY	Flash Partition Boundary in Descriptor is outside flash address space
0x10	KERNEL_CHIPSET_DESCRIPTOR_ACCE_SS_DENIED	ME region cannot access the Chipset Descriptor Region
0x11	KERNEL_UNSUPPORTED_FLASH_CONFIGURATION	Required VSCC values for flash parts do not match
0x12	KERNEL_STATUS_LOAD_BUP	uKernel loads BUP
0x13	EXCEPTION_ARC	State Info specified by the Exception Handler
0x14	EXCEPTION_BACKBONE	State Info specified by the Exception Handler
0x15	EXCEPTION_NON_COMP	State Info specified by the Exception Handler
0x16	EXCEPTION_PCIM	State Info specified by the Exception Handler
0x17	EXCEPTION_WD_TIMER	State Info specified by the Exception Handler
0x18	EXCEPTION_SUPERTASK_CODE_CACHE	State Info specified by the Exception Handler
0x19	EXCEPTION_SUPERTASK_DATA_CACHE	State Info specified by the Exception Handler
0x1A	EXCEPTION_SUPERTASK_AUX	State Info specified by the Exception Handler
0x1B	EXCEPTION_SUPERTASK_ARC	State Info specified by the Exception Handler
0x1C	EXCEPTION_SUPERTASK_CODE	State Info specified by the Exception Handler

Value	Name	Description
0x1D	KERNEL_PM_GO_DS3_RCVD	Received DS3 entry MSI from PMC

28.6.2.14.4 HECI1_CSE_GS1_PHASE_BUP

Value	Name	Description
0x00	BUP_STATUS_BEGIN	Initialization starts
0x01	BUP_STATUS_DISABLE_HOST_WAKE_EVENT	Disable the host wake event
0x02	BUP_STATUS_CLOCK_GATING_ENABLED	Enabling clock gating for CSE
0x03	BUP_STATUS_HOST_PM_HANDSHAKE_ENABLED	Enabling PM ME handshaking
0x04	BUP_STATUS_FLOW_DETERMINATION	Flow determination start process
0x05	BUP_STATUS_PMC_PATCHING	PMC patching process
0x06	BUP_STATUS_GET_FLASH_VSCC	Get the flash VSCC parameters
0x07	BUP_STATUS_SET_FLASH_VSCC	Set (program) the flash VSCC registers
0x08	BUP_STATUS_VSCC_FAILURE	Error reading/matching the VSCC table in the descriptor
0x09	BUP_STATUS_EFFS_INITIALIZATION	Initialize EFS (overlay and SDM memory management)
0x0A	BUP_STATUS_CHECK_CSE_STRAP_DISABLED	Check to see if straps say CSE DISABLED
0x0B	BUP_STATUS_T34_MISSING	Timeout waiting for T34
0x0C	BUP_STATUS_CHECK_STRAP_DISABLED	Check to see if effs say CSE DISABLED
0x0D	BUP_STATUS_CHECK_FLASH_OVERRIDE	Possibly handle BUP manufacturing override strap
0x0E	BUP_STATUS_CHECK_CSE_POLICY_DISABLED	Check to see if effs say CSE DISABLED
0x0F	BUP_STATUS_PKTPM_INITIALIZATION	Initialize PKTPM
0x10	BUP_STATUS_BLOB_INITIALIZATION	Initialize MiniBLOB
0x11	BUP_STATUS_CM3	Bringup in CM3
0x12	BUP_STATUS_CM0	Bringup in CM0
0x13	BUP_STATUS_FLOW_ERROR	Flow detection error
0x14	BUP_STATUS_CM3_CLOCK_SWITCH	CM3 clock switching
0x15	BUP_STATUS_CM3_CLOCK_SWITCH_ERROR	CM3 clock switching error
0x16	BUP_STATUS_CM3_FLASH_PAGING	CM3 flash paging flow
0x17	BUP_STATUS_CM3_ICV_RECOVERY	CM3 ICV recovery flow



Value	Name	Description
0x18	BUP_STATUS_CMO_LOAD_KERNEL	CM3 kernel load
0x19	BUP_STATUS_CMO_HOST_PREP	CM0 Host prep sequence
0x1A	BUP_STATUS_CMO_SKIP_HOST_PREP	CM0 Host skip prep sequence
0x1B	BUP_STATUS_CMO_ICC_PROGRAMMING	ICC programming
0x1C	BUP_STATUS_CMO_T34_ERROR	T34 missing - cannot program ICC
0x1D	BUP_STATUS_CMO_FLEX_SKU	FLEX SKU programing
0x1E	BUP_STATUS_CMO TPM_START	TPM start (interface unisolation
0x1F	BUP_STATUS_CMO_DID_WAIT	Waiting for DID BIOS message
0x20	BUP_STATUS_CMO_DID_ERROR	Waiting for DID BIOS message failure
0x21	BUP_STATUS_CMO_DID_NOMEM	DID reported no error.
0x22	BUP_STATUS_CMO_UMA_ENABLE	Enabling UMA
0x23	BUP_STATUS_CMO_UMA_ENABLE_ERROR	Enabling UMA error
0x24	BUP_STATUS_CMO_DID_ACK	Sending DID Ack to BIOS
0x25	BUP_STATUS_CMO_DID_ACK_ERROR	Sending DID Ack to BIOS error
0x26	BUP_STATUS_CMO_CLOCK_SWITCH	Switching clocks in M0
0x27	BUP_STATUS_CMO_CLOCK_SWITCH_ERROR	Switching clocks in M0 error
0x28	BUP_STATUS_CMO_TEMP_DISABLE	CSE in temp disable
0x29	BUP_STATUS_CMO_TEMP_DISABLE_ERROR	CSE in temp disable – error
0x2A	BUP_STATUS_CMO_TEMP_DISABLE_UMA_ENABLE	CSE in temp disable - exiting and UMA enabling
0x2B	BUP_STATUS_CMO_IPK_CHECK	CSE IPK check
0x2C	BUP_STATUS_CMO_IPK_RECREATION	CSE IPK recreation
0x2D	BUP_STATUS_CMO_IPK_RECREATION_ERROR	CSE IPK recreation error
0x2E	BUP_STATUS_CMO_UMA_VALIDATION	CSE UMA validation for resume
0x2F	BUP_STATUS_CMO_UMA_VALIDATION_ERROR	CSE UMA validation for resume error
0x30	BUP_STATUS_CMO_UMA_VALIDATION_IPK	CSE UMA validation for resume error, so IPK recreation
0x31	BUP_STATUS_CMO_PKVENOM_START	CM0 PK VENOM start
0x32	BUP_STATUS_CMO_LOAD_IBLS	CM0 load IBLs
0x33	BUP_STATUS_CMO_PK_FTPM_INIT	CM0 PK FTPM init phase
0x34	BUP_STATUS_CMO_PK_FTPM_ABORT	CM0 PK FTPM Abort

Value	Name	Description
0x35	BUP_STATUS_HALT_UPON_FIPS_BUP_ERR	FIPS halt - BUP self-test error
0x36	BUP_STATUS_HALT_UPON_FIPS_CRYPT_O_DRV_ERR	FIPS halt - CRYPTO_DRV self-test error
0x37	BUP_STATUS_HALT_UPON_FIPS_TLS_ER	FIPS halt - TLS self-test error
0x38	BUP_STATUS_HALT_UPON_FIPS_DT_ER	FIPS halt - DT self-test error
0x39	BUP_STATUS_HALT_UPON_FIPS_UNKNOW_N_ERR	FIPS halt - DT self-test error
0x3A	BUP_STATUS_CMO_MEMORY_ACCESS_RANGE_ERR	Error enabling memory acces range
0x3B	BUP_STATUS_CSE_RESET_LIMIT_ERR	ME reset limit reached
0x3C	BUP_STATUS_CSE_RESET_ENTER_RECOVERY	Two ME resets within short time detected, enter recovery
0x3D	BUP_STATUS_HALT_UPON_UNKNOWN_ERROR	Unknown halt reason detected, halt ME
0x3E	BUP_STATUS_VALIDATE_NFT	Validating GLUT and NFT manifest
0x3F	BUP_STATUS_READ_FIXED_DATA	Reading kernel fixed data from NVAR
0x40	BUP_STATUS_READ_ICC_DATA	Reading ICC data
0x41	BUP_STATUS_ZERO_UMA	Zeroing out UMA
0x42	BUP_STATUS_HALT_UPON_SKU_ERROR	Full Fw sku running on Ignition HW sku
0x43	BUP_STATUS_DERIVE_CHIPSET_KEY_ERROR	Error when deriving chipset key
0x44	BUP_STATUS_HOST_ERROR	Bad BIOS, CPU DOA, CPU Missing
0x45	Reserved	Reserved
0x46	BUP_STATUS_FTP_LOAD_ERROR	Failure in loading FTP
0x47	BUP_STATUS_MFG_CM_RST	CSE halted in BUP from Mfg ME reset
0x48	BUP_STATUS_MPR_VIOLATION_CM_RST	CSE was reset because of MPR protection violation
0x49	BUP_STATUS_ICC_START_POLL_BEGIN	START_ICC_CONFIG polling begin
0x4A	BUP_STATUS_ICC_START_POLL_END	START_ICC_CONFIG polling end
0x4B	BUP_STATUS_HOBIT_SET	set HOBIT
0x4C	BUP_STATUS_POLL_CPURST_DEASSERT_BEGIN	CPU_RST_DONE polling begin
0x4D	BUP_STATUS_CPURST_DEASSERT_DONE	CPU_RST_DONE polling end



Value	Name	Description
0x4E	BUP_STATUS_DID_POLL_BEGIN	Dram Init Done polling begin
0x4F	BUP_STATUS_DID_RECVD	Dram Init Done polling end
0x50	BUP_STATUS_ZERO_UMA_BEGIN	
0x51	BUP_STATUS_ZERO_UMA_DONE	
0x52	BUP_STATUS_DID_ACK_SENT	Dram Init Done Ack sent to BIOS
0x53	BUP_STATUS_ICC_REQUEST_GLOBAL_RESET	ICC requested a global reset after DID timeout
0x54	BUP_STATUS_MTP_CLOCK_FREQ_CHECK	Perform ROSC clock frequency check
0x55	BUP_STATUS_MTP_CLOCK_FREQ_CHECK_FAIL	Clock check failed
0x56	BUP_STATUS_CPURST_DEASSERT_FAIL	CPU_RESET_DONE_ACK not received
0x57	BUP_STATUS_WORKWEEK_YEAR_FAIL	FW build does not support the WW and/or Year specified in the fuses
0x58	BUP_STATUS_ULV_CHECK_START	ULV PCH check in progress
0x59	BUP_STATUS_ULV_CHECK_FAIL	ULV PCH not paired with LV/ULV CPU
0x5A	BUP_STATUS_VDM_HW_FAIL	VDM handshake failure
0x5B	Reserved	Reserved
0x5C	Reserved	Reserved
0x5D	BUP_STATUS_FFS_ENTRY	FFS entry
0x5E	BUP_STATUS_FFS_EXIT	FFS exit
0x60	BUP_STATUS_TRNG_ERROR	Removed in future, use DRNG and no TRNG in ME9
0x60	BUP_STATUS_DRNG_ERROR	Error when getting a random number from DRNG
0x61	BUP_STATUS_GET_SP_CANARY	Find stack protection canary from NVAR or TRNG
0x62	BUP_STATUS_VDM_GET_SID	VDM Get SID Message in progress
0x63	BUP_STATUS_VLB_WAIT	Wait for CPU to issue VLB authentication response
0x64	BUP_STATUS_GPDMA_MEMORY_ACCESS_CMIRST	ME Reset due to invalid access of host memory by NP
0x65	BUP_STATUS_PCH_MISMATCH	PCH HW type Mismatch versus what was emulated using FIT
0x66	BUP_STATUS_MB_P_WRITE	MBP written to HECI buffer
0x67	BUP_STATUS_FFS_EXIT_ERROR	FW expects FFS exit but BIOS did not set bit in DID

Value	Name	Description
0x68	BUP_STATUS_DID_TIMEOUT_FORCE_SH_UTDOWN	fatal error handler after PM driver saw extended D_I_D timeout with AT enabled
0x69	BUP_STATUS_PMDRV_EXT_DID_TIMEOUT	fatal error handler after PM driver saw extended D_I_D timeout with AT enabled
0x6A	BUP_STATUS_PMDRV_DID_TIMEOUT	fatal error handler after PM driver saw D_I_D timeout with AT enabled
0x6B	Reserved	Reserved
0x6C	BUP_STATUS_DEEP_S3_EXIT	Exiting DeepS3
0x6D	BUP_STATUS_DEEP_S3_EXIT_ERROR	Error exiting DeepS3
0x6E	BUP_STATUS_DRNG_BIST_INCOMPLETE_CMRST	
0x6F	BUP_STATUS_DRNG_BIST_ES_KAT_FAILURE_CMRST	
0x70	BUP_STATUS_DRNG_BIST_INTEGRITY_FAILURE_CMRST	
0x71	BUP_STATUS_DRNG_FUSE_ERROR_CMRST	
0x72	BUP_STATUS_DRNG_TIMEOUT_CMRST	
0x73	BUP_STATUS_DID_ACK_REQ_GRST	
0x74	BUP_STATUS_DID_ACK_REQ_PCR	
0x75	BUP_STATUS_DID_ACK_REQ_NPCR	
0x76	BUP_STATUS_SAFE_MODE_ENTRY	
0x77	BUP_STATUS_RECOVERY_ENTRY_FTP_BAD	
0x78	BUP_STATUS_RECOVERY_ENTRY_NFTP_BAD	
0x79	BUP_STATUS_START_FIRST_BOOT_ME_HASH	
0x7A	BUP_STATUS_END_OF_FIRST_BOOT_ME_HASH	
0x7B	BUP_STATUS_PCH_ID_MISMATCH	PCH ID (aka UMCHID) Mismatch for production platform only Saved in flash UMCHID does not match UMCHID read from ROM bist data.
0x7C	BUP_STATUS_GRST_AT_REQUEST	
0x7D	BUP_STATUS_GRST_AT_TIMER_EXPIRE	
0x7E	BUP_STATUS_CLINK_FATAL_ERROR_CM_RST	



Value	Name	Description
0x7F	BUP_STATUS_SECURE_BOOT_FAILURE	
0x80	BUP_STATUS_EXCEPTION_RST_PBO	
0x81	BUP_STATUS_SECURE_BOOT_SM_FAILURE	
0x82	BUP_STATUS_SECURE_BOOT_EN_PCH_PLUGGED_IN	
0x83	BUP_STATUS_MEBX_INVOCATION_REQUESTED	
0x84	BUP_STATUS_CMO_MKHI_HANDLER_START	
0x85	BUP_STATUS_CMO_MKHI_HANDLER_STOP	
0x86	BUP_STATUS_CMO_MB_P_WRITE_SUCCESS	
0x87	BUP_STATUS_CMO_MB_P_WRITE_ERROR	
0x88	BUP_STATUS_HOST_BOOT_PREP_FAIL	
0x89	BUP_STATUS_HECI_LINK_RESET_START	
0x8A	BUP_STATUS_HECI_LINK_RESET_DONE	
0x8B	BUP_STATUS_UNSUPPORTED_PROD_PCH	
0x8C	BUP_STATUS_UNSUPPORTED_SERVER_PCH	
0x8D	BUP_STATUS_XEON_CPU_AND_NON_SERVER_WS_PCH	
0x8E	BUP_STATUS_LP_AND_H_PCH_MISMATCH	
0x8F	BUP_STATUS_NON_PV_FW_ON_REVENE_E_HW_FAIL	
0x90	BUP_STATUS_DT_H_CPU_AND_MB_PCH	
0x91	BUP_STATUS_MBL_H_CPU_AND_DT_PCH	

28.6.2.14.5HECI1_CSE_GS1_PHASE_LOAD

Value	Name	Description
0x00	LOAD_STATUS_BEGIN	Initialization starts

28.6.2.14.6HECI1_CSE_GS1_PHASE_IDLM

Value	Name	Description
0x00	IDLM_START	Entry into IDLM Module
0x01	IDLM_UMCHID_NOT_PROGRAMMED	The platform's UMCHID not created
0x02	IDLM_PLATFORM_UNAUTHORIZED	The platform's UMCHID was not in the list
0x03	IDLM_FAILED_TO_VALIDATE_FPT	Failed to validate the FPT
0x04	IDLM_CODE_PARTITION_NOT_FOUND	FTPR partition not found in FW image
0x05	IDLM_FAILED_TO_VALIDATE_BUP_MANIFEST	Failed to validate the FTPR manifest in FW image
0x06	IDLM_FAILED_TO_ALLOCATE_PAGES	Failed to allocate pages
0x07	IDLM_BUP_MODULE_NOT_FOUND	BUP module not found in FW image
0x08	IDLM_LOADING_BRINGUP	Loading BringUp
0x09	IDLM_PCH_GEN_UNAUTHORIZED	GDID reported a non PPT PCH Generation
0x0A	IDLM_BACKBONE_EXCEPTION	Backbone exception found
0x0B	IDLM_PIDS_PARTITION_NOT_FOUND	Failed to find PIDS manifest marker
0x0C		
0x0D	IDLM_PIDS_MODULE_NOT_FOUND	PIDs module not found in IDLM image
0x0E	IDLM_FAILED_TO_LOAD_PID_MODULE	Failed to load PIDS module
0x0F	IDLM_EXCEEDED_MAX_NUM_PIDS	Exceeded max number of PIDs
0x10	IDLM_FAILED_TO_VALIDATE_PID_MANIFEST	Failed to validate PID manifest
0x11	IDLM_FAILED_TO_VALIDATE_PID_MARKER	Failed to validate PID marker
0x12	IDLM_ERROR_DURING_HASH_CREATION	Failure during Hash creation
0x13	IDLM_MFGFUSES_NOT_PROGRAMMED	The platform's Mfg Fuses are unavailable
0x14	IDLM_FAILURE_DURING_KEY_CREATION	Failure during key generation
0x15	IDLM_PRAMV_ZEROING_FAIL_TO_START	PRAM-V zeroing took an unreasonable amount of time

28.6.2.14.7HECI1_CSE_GS1_PHASE_HOSTCOMM

Value	Name	Description
0x00	HOSTCOMM_STATUS_BEGIN	Initialization starts
0x01		



Value	Name	Description
0x02		
0x03		
0x04		
0x05		
0x06		
0x07		
0x08		
0x09		
0x05		
0x0A		
0x0B		
0x0C		
0x0D		
0x0E		
0x0F		
0x10		
0x11		
0x12		
0x13		
0x14		
0x15		

28.6.2.14.8HECI1_CSE_GS1_PHASE_FWUPDATE

Value	Name	Description
0x00		
0x01		
0x02		
0x03		
0x04		
0x05		
0x06		
0x07		
0x08		
0x09		

Value	Name	Description
0x05		
0x0A		
0x0B		
0x0C		
0x0D		
0x0E		
0x0F		
0x10		
0x11		
0x12		
0x13		
0x14		
0x15		

28.6.2.14.9HECI1_CSE_GS1_PHASE_MAESTRO

Value	Name	Description
0x00		
0x01		
0x02		
0x03	MAESTRO_PM_GO_S3_RCVD	Received S3 entry MSI from PMC
0x04	MAESTRO_PM_GO_S4_RCVD	Received S4 entry MSI from PMC
0x05	MAESTRO_PM_GO_S5_RCVD	Received S5 entry MSI from PMC
0x06	MAESTRO_PM_POWER_DOWN_RCVD	Received UPD entry MSI from PMC
0x07	MAESTRO_PM_PWR_CYCLE_RST_RCVD	Received PCR entry MSI from PMC
0x08	MAESTRO_PM_NON_PWR_CYCLE_RST_RCVD	Received NPCR entry MSI from PMC



Value	Name	Description
0x09	MAESTRO_PM_HOST_WAKE_RCVD	Received host wake MSI from PMC
0x0A	MAESTRO_PM_PWR_SRC_AC	power source AC
0x0B	MAESTRO_PM_PWR_SRC_DC	power source DC
0x0C	MAESTRO_CMO_PG_ENTRY	
0x0D	MAESTRO_CMO_PG_EXIT	
0x0E	MAESTRO_CM3_PG_ENTRY	
0x0F	MAESTRO_CMOFF_ENTRY	
0x10	MAESTRO_CMO_ENTRY_COMPLETE_IDLE	
0x11	MAESTRO_CMO_ENTRY_COMPLETE_START	
0x12	MAESTRO_CMO_ENTRY_COMPLETE_PUBLISHING	
0x13	MAESTRO_CMO_ENTRY_BEGIN_CMOFF_EXIT	
0x14	MAESTRO_CMO_ENTRY_BEGIN_LIVE	
0x15		
0x16		
0x17		
0x18		
0x19		
0x1A		
0x1B		
0x1C		
0x1D		
0x1E		
0x1F		
0x20	MAESTRO_CMO_EXIT_BEGIN	
0x21	MAESTRO_CMO_EXIT_BEGIN_IDLE_CMO_PG	
0x22	MAESTRO_CMO_EXIT_BEGIN_GO_SX_CMOFF	
0x23	MAESTRO_CMO_EXIT_BEGIN_GO_SX_CM3	
0x24	MAESTRO_CMO_EXIT_BEGIN_RST_WARN_CMOFF	
0x25	MAESTRO_CMO_EXIT_BEGIN_RST_WARN_CMO_NU	
0x26	MAESTRO_CMO_EXIT_BEGIN_RST_WARN_CM3	
0x27	MAESTRO_CMO_EXIT_BEGIN_GLB_RST_REQ_CMOFF	

Value	Name	Description
0x28	MAESTRO_CMO_EXIT_BEGIN_CSE_RST_REQ_CMOFF	
0x29		
0x2A		
0x2B		
0x2C		
0x2D		
0x2E		
0x2F		
0x30	MAESTRO_CM3_ENTRY_COMPLETE_IDLE	
0x31	MAESTRO_CM3_ENTRY_COMPLETE_START	
0x32	MAESTRO_CM3_ENTRY_COMPLETE_PUBLISHING	
0x33	MAESTRO_CM3_ENTRY_BEGIN_CMOFF_EXIT	
0x34	MAESTRO_CM3_ENTRY_BEGIN_LIVE	
0x35	MAESTRO_CM3_ENTRY_BEGIN_TRY_ABORT	
0x36	MAESTRO_CM3_ENTRY_BEGIN_ABORT_SUCCESS	
0x37	MAESTRO_CM3_ENTRY_BEGIN_ABORT_FAIL_WAIT_TC	
0x38		
0x39		
0x3A		
0x3B		
0x3C		
0x3D		
0x3F		
0x40	MAESTRO_CM3_EXIT_BEGIN	
0x41	MAESTRO_CM3_EXIT_BEGIN_IDLE_CM3_PG	
0x42	MAESTRO_CM3_EXIT_BEGIN_HOST_WAKE_CMO_NU	
0x43	MAESTRO_CM3_EXIT_BEGIN_AC_OR_POLICY_CMOFF	
0x44	MAESTRO_CM3_EXIT_BEGIN_GLB_RST_REQ_CMOFF	
0x45	MAESTRO_CM3_EXIT_BEGIN_CSE_RST_REQ_CMOFF	
0x46		
0x47		
0x48		
0x49		
0x4A	MAESTRO_CM3_EXIT_COMPLETE_CMOFF	



Value	Name	Description
0x4B	MAESTRO_CMO3_EXIT_COMPLETE_CMO_NU	
0x4C		
0x4D		
0x4E		
0x4F		
0x50	MAESTRO_CMO_NU_ENTRY_COMPLETE	
0x51		
0x52	MAESTRO_CMO_NU_ENTRY_COMPLETE_PUBLISHING	
0x53	MAESTRO_CMO_NU_ENTRY_BEGIN_CMOFF_EXIT	
0x54	MAESTRO_CMO_NU_ENTRY_BEGIN_LIVE	
0x55	MAESTRO_CMO_NU_ENTRY_BEGIN_ICC_CONFIG_START	
0x56	MAESTRO_CMO_NU_ENTRY_BEGIN_BUP_PREP_HOST_BOOT	
0x57	MAESTRO_CMO_NU_ENTRY_BEGIN_SET_HOST_BOOT_PREP_DONE	
0x58	MAESTRO_CMO_NU_ENTRY_BEGIN_CPU_RST_DONE_ACK	
0x59	MAESTRO_CMO_NU_ENTRY_BEGIN_WAITING_ON_DID_AND_TC	
0x5A	MAESTRO_CMO_NU_ENTRY_BEGIN_WAITING_ON_DID	
0x5B	MAESTRO_CMO_NU_ENTRY_BEGIN_WAITING_ON_TC	
0x5C		
0x5D		
0x5E		
0x5F		
0x60	MAESTRO_CMO_NU_EXIT_BEGIN	
0x61	MAESTRO_CMO_NU_EXIT_BEGIN_CMO	
0x62	MAESTRO_CMO_NU_EXIT_BEGIN_GO_SX_CMOFF	
0x63	MAESTRO_CMO_NU_EXIT_BEGIN_GO_SX_CM3	
0x64	MAESTRO_CMO_NU_EXIT_BEGIN_RST_WARN_CMOFF	
0x65	MAESTRO_CMO_NU_EXIT_BEGIN_RST_WARN_CMO_NU	
0x66	MAESTRO_CMO_NU_EXIT_BEGIN_GLB_RST_REQ_CMOFF	
0x67	MAESTRO_CMO_NU_EXIT_BEGIN_CSE_RST_REQ_CMOFF	
0x68	MAESTRO_CMO_NU_EXIT_BEGIN_TEMP_DISABLE_CMOFF	
0x69		
0x6A	MAESTRO_CMO_NU_EXIT_COMPLETE_CMOFF	
0x6B	MAESTRO_CMO_NU_EXIT_COMPLETE_CMO_NU	
0x6C	MAESTRO_CMO_NU_EXIT_COMPLETE_CM3	

Value	Name	Description
0x6D	MAESTRO_CMO_NU_EXIT_COMPLETE_CMO	
0x6E		
0x6F		

28.6.2.15 HECI1_CSE_GS1_PM

28.6.2.16 HECI1_CSE_GS1_PHASE

Current phase of operation.

Value	Name	Description
0	HECI1_CSE_GS1_PHASE_ROM	State Info specified by the ROM
1	HECI1_CSE_GS1_PHASE_RBE	State Info specified by the RBE
2	HECI1_CSE_GS1_PHASE_UKERNEL	State Info specified by the Privilege micro-kernel
3	HECI1_CSE_GS1_PHASE_BUP	State Info specified by the MemInit/BRINGUP
4	HECI1_CSE_GS1_PHASE_LOAD	State Info specified by the Recovery Mini-Loader
5	HECI1_CSE_GS1_PHASE_IDLM	State Info specified by the IDLM
6	HECI1_CSE_GS1_PHASE_HOSTCOMM	State Info specified by the HOSTCOMM module
7	HECI1_CSE_GS1_PHASE_FWUPDATE	State Info specified by the FWUPDATE module
8	HECI1_CSE_GS1_PHASE_MAESTRO	State Info specified by the MAESTRO

28.7 Intel® Management Engine Kernel Host Interface (MKHI) Messages

All the messages from Host to TXE Kernel and vice versa have been consolidated under MKHI messaging. This is done to stream line messaging between Host and TXE Kernel. All the messages go through the MKHI dispatcher in TXE Kernel and the dispatcher in turn forwards the message to the appropriate kernel component based



on group ID. The following section defines the generic data types and structure used in each MKHI message communication.

MKHI messages require TXE Kernel to be up running. The following MKHI messages do not need to check HECL1_HFS.FWInitComplete bit value before being sent by IA FW.

28.7.1 MKHI Header Generic Types and Structures

Table 28-30. Definition for MKHI Header

Each MKHI Request and Response has a header.

Size Bytes	Field Name	Field Values	Description
1	GroupId	0-0xFF	The MKHI group ID that handles the command.
1	CmdId	0-0xFF	BIT[7:0] – CmdID BIT[7] – Is Response CmdID is unique for each GroupId Is Response bit is set on ACK.
1	Reserved	0x0	Reserved
1	Result	0-0xFF	Result set on ACK to indicate result. 0 – Success X – Failure defined per GroupId/CmdId pair.

28.7.2 MKHI – Core BIOS Group

```
#define MKHI_CBM_GROUP_ID 0
```

28.7.3 MKHI – FW Caps Group

```
#define MKHI_FWCAPS_GROUP_ID 0x3
```

28.7.3.1 GET RULE

Availability

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	No	Yes	
Post Manufacturing	No	Yes	No	Yes	
Recovery	No	No	No	No	Not available.

**Handler**

CMO Process	BUP Process	MBP Entry
Policy	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00000203	MKHI Group ID GROUP ID 0x03 Command 0x02
2	RuleTypeId		Rule ID
1	FeatureId		Feature ID
1	Reserved		Reserved

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX008303 (XX – see result code).	Result Code
2	RuleTypeId		Rule ID
1	FeatureId		Feature ID
1	Reserved		Reserved
1	DataLen		The length of rule date returned.
DataLen	RuleData		The rule data

28.7.3.2 SET RULE**Availability**

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	No	Yes	
Post Manufacturing	No	Yes	No	Yes	
Recovery	No	No	No	No	Not available.

**Handler**

CMO Process	BUP Process	MBP Entry
Policy	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00000303	MKHI Group ID GROUP ID 0x03 Command 0x03
2	RuleTypeId		Rule ID
1	FeatureId		Feature ID
1	Reserved		Reserved
1	DataLen		Rule Data length
DataLen	RuleData		The rule data to set.

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX008303 (XX – see result code).	Result Code
2	RuleTypeId		Rule ID
1	FeatureId		Feature ID
1	Reserved		Reserved
DataLen	RuleData		The rule data

Result Code

Value	Code	Description
0x00	OK	Manifest successfully verified
0x01	OK_CHUNK	Successful chunk process
0x05	SIZE_ERROR	Command size does not match expected value.
0x0B	EMANHDR	Manifest Header Invalid
0x0C	EINVSIG	Manifest sig invalid
0x8D	STATUS_INVALID_COMMAND	Command not supported.
0xFE	RETRY	Not CMO T.C. retry command.
0xFF	GEN_ERROR	General error with manifest verify.

28.7.4 MKHI – MCA Group

```
#define MKHI_RPMC_GROUP_ID 0xA
```

28.7.4.1 CORE BIOS DONE

Availability

	Before DID	Post DID	Post EOP	Post EOS	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	Yes	
Post Manufacturing	No	Yes	Yes	Yes	Yes	
Recovery	No	Yes	Yes	Yes	Yes	

Handler

CMO Process	BUP Process	MBP Entry
MCA	Yes	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x0000050A	MKHI Group ID GROUP ID 0x0A Command 0x05

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX00850A (XX – see result code).	Result Code

Value	Code	Description
0x00	OK	Command completed successfully.
0xFF	GEN_ERROR	General error with command.

28.7.4.2 READ DATA

Availability

	Before DID	Post DID	Post CBD	Post EOP	Post EOS	Before FID	Notes
Manufacturing	No	Yes	Yes	No	No	Yes	Not locked by CBD.



	Before DID	Post DID	Post CBD	Post EOP	Post EOS	Before FID	Notes
Post Manufacturing	No	Yes	Yes	No	No	Yes	
Recovery	No	Yes	Yes	No	No	Yes	

Handler

CMO Process	BUP Process	MBP Entry
MCA	Yes	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00000C0A	MKHI Group ID GROUP ID 0x0A Command 0x0C
25	FilePath	ASCII	NULL terminated string in format <dirName/Filename>
4	Offset		Offset in file in bytes.
4	Size		Size in bytes to read
4	DstAddressLower		PA of lower 32 bits of destination buffer for CSE to copy to.
4	DstAddressUpper		PA of upper 32 bits of destination buffer for CSE to copy to.

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX008C0A (XX – see result code).	Result Code
4	DataSize		Size of written data in bytes.

Value	Code	Description
0x00	OK	Command completed successfully.
0x01	BIOS_HECI_STATUS_INVALID_PARAM	
0x02	BIOS_HECI_STATUS_FILE_NOT_FOUND	
0x03	BIOS_HECI_STATUS_AFTER_EOS	

Value	Code	Description
0x04	BIOS_HECI_STATUS_DIR_LOCKED	
0x05	BIOS_HECI_STATUS_NVM_INACCESSIBLE	
0x06	BIOS_HECI_STATUS_TOO_MANY_DIR	
0x07	BIOS_HECI_STATUS_COUNTER_INVALID	
0x08	BIOS_HECI_STATUS_SIGN_INVALID	
0x09	BIOS_HECI_STATUS_POST_EOP	
0x0A	BIOS_HECI_STATUS_POST_CORE_BIOS_DONE	
0x0B	BIOS_HECI_STATUS_NO_QUOTA	
0x0F	BIOS_HECI_STATUS_ERROR	
0xFF	GEN_ERROR	General error with command.

Details

BUP

- BUP - storage performs read.
- BUP – storage will use the BUP ATT window and BUP – busdrv to program the ATT window.

MCA

- MCA – Call BUP exported API and re-use ATT BUP window.

BUP MKHI API

- The exported API will be called by MCA and provide the message handling on behalf of MCA.
- BUP will track EOP and cut off the command when EOP has been received.

28.7.4.3 WRITE DATA

Availability

	Before DID	Post DID	Post CBD	Post EOP	Post EOS	Before FID	Notes
Manufacturing	No	Yes	No	No	No	Yes	Not available after CBD.
Post Manufacturing	No	Yes	No	No	No	Yes	Not available after CBD.
Recovery	No	No	No	No	No	No	Not available.

Handler

CMO Process	BUP Process	MBP Entry
MCA	No	

**MKHI Request**

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00000D0A	MKHI Group ID GROUP ID 0xA Command 0xD
25	FilePath	ASCII	NULL terminated string in format <dirName/Filename>
4	Offset		Offset in file in bytes.
4	Size		Size in bytes to read
4	Truncate	0-1	0 – Do not set posix truncate flag when opening file. 1 – Set posix truncate flag when opening file.
4	SrcAddressLower		PA of lower 32 bits of source buffer for CSE to copy to.
4	SrcAddressUpper		PA of upper 32 bits of source buffer for CSE to copy to.

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX008D0A (XX – see result code).	Result Code

Result Code

Value	Code	Description
0x00	OK	Command completed successfully.
0x01	BIOS_HECI_STATUS_INVALID_PARAM	
0x02	BIOS_HECI_STATUS_FILE_NOT_FOUND	
0x03	BIOS_HECI_STATUS_AFTER_EOS	
0x04	BIOS_HECI_STATUS_DIR_LOCKED	
0x05	BIOS_HECI_STATUS_NVM_INACCESSIBLE	
0x06	BIOS_HECI_STATUS_TOO_MANY_DIR	
0x07	BIOS_HECI_STATUS_COUNTER_INVALID	
0x08	BIOS_HECI_STATUS_SIGN_INVALID	
0x09	BIOS_HECI_STATUS_POST_EOP	
0x0A	BIOS_HECI_STATUS_POST_CORE_BIOS_DONE	
0x0B	BIOS_HECI_STATUS_NO_QUOTA	
0x0F	BIOS_HECI_STATUS_ERROR	



0xFF	GEN_ERROR	General error with command.
------	-----------	-----------------------------

Details

MCA

- MCA – Will handle.

28.7.4.4 GET FILE SIZE**Availability**

	Before DID	Post DID	Post CBD	Post EOP	Post EOS	Before FID	Notes
Manufacturing	No	Yes	Yes	No	No	Yes	Not locked by CBD.
Post Manufacturing	No	Yes	Yes	No	No	Yes	
Recovery	No	Yes	Yes	No	No	Yes	

Handler

CMO Process	BUP Process	MBP Entry
MCA	Yes	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00000EOA	MKHI Group ID GROUP ID 0x0A Command 0x0E
25	FilePath	ASCII	NULL terminated string in format <dirName/Filename>

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX008EOA (XX – see result code).	Result Code
4	DataSize		Size of the file in bytes.

**Result Code**

Value	Code	Description
0x00	OK	Command completed successfully.
0x01	BIOS_HECI_STATUS_INVALID_PARAM	
0x02	BIOS_HECI_STATUS_FILE_NOT_FOUND	
0x03	BIOS_HECI_STATUS_AFTER_EOS	
0x04	BIOS_HECI_STATUS_DIR_LOCKED	
0x05	BIOS_HECI_STATUS_NVM_INACCESSIBLE	
0x06	BIOS_HECI_STATUS_TOO_MANY_DIR	
0x07	BIOS_HECI_STATUS_COUNTER_INVALID	
0x08	BIOS_HECI_STATUS_SIGN_INVALID	
0x09	BIOS_HECI_STATUS_POST_EOP	
0x0A	BIOS_HECI_STATUS_POST_CORE_BIOS_DONE	
0x0B	BIOS_HECI_STATUS_NO_QUOTA	
0x0F	BIOS_HECI_STATUS_ERROR	
0xFF	GEN_ERROR	General error with command.

Details

BUP

- BUP - storage performs read.
- BUP – storage will use the BUP ATT window and BUP – busdrv to program the ATT window.

MCA

- MCA – Call BUP exported API and re-use ATT BUP window.

BUP MKHI API

- The exported API will be called by MCA and provide the message handling on behalf of MCA.
- BUP will track EOP and cut off the command when EOP has been received.

28.7.4.5 REQUEST DEVICE OWNERSHIP**Availability**

	Before DID	Post DID	Post CBD	Post EOP	Post EOS	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	No	Yes	Not locked by CBD.
Post Manufacturing	No	Yes	Yes	Yes	No	Yes	



	Before DID	Post DID	Post CBD	Post EOP	Post EOS	Before FID	Notes
Recovery	No	Yes	Yes	Yes	No	Yes	

Before DID		After EOP	After EOM	Recovery	Other
No		No	Yes	No	MBP entry for recovery.

Handler

CMO Process	BUP Process	MBP Entry
MCA	Yes	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00000FOA	MKHI Group ID GROUP ID 0x0A Command 0x0F

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX008FOA (XX – see result code).	Result Code

Result Code

Value	Code	Description
0x00	OK	Command completed successfully.
0x01	BIOS_HECI_STATUS_INVALID_PARAM	
0x02	BIOS_HECI_STATUS_FILE_NOT_FOUND	
0x03	BIOS_HECI_STATUS_AFTER_EOS	
0x04	BIOS_HECI_STATUS_DIR_LOCKED	
0x05	BIOS_HECI_STATUS_NVM_INACCESSIBLE	
0x06	BIOS_HECI_STATUS_TOO_MANY_DIR	
0x07	BIOS_HECI_STATUS_COUNTER_INVALID	
0x08	BIOS_HECI_STATUS_SIGN_INVALID	



Value	Code	Description
0x09	BIOS_HECI_STATUS_POST_EOP	
0x0A	BIOS_HECI_STATUS_POST_CORE_BIOS_DONE	
0x0B	BIOS_HECI_STATUS_NO_QUOTA	
0x0F	BIOS_HECI_STATUS_ERROR	
0xFF	GEN_ERROR	General error with command.

Details

BUP will handle by calling API in Storage IBL. For recovery BUP will handle directly.

28.7.4.6 Lock HOST SW DIRECTORY WRITES

Availability

	Before DID	Post DID	Post CBD	Post EOP	Post EOS	Before FID	Notes
Manufacturing	No	Yes	Yes	No	No	Yes	Not locked by CBD.
Post Manufacturing	No	Yes	Yes	No	No	Yes	
Recovery	No	No	Yes	No	No	No	Not available.

Handler

CMO Process	BUP Process	MBP Entry
MCA	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x0000100A	MKHI Group ID GROUP ID 0x0A Command 0x10
25	FilePath	ASCII	NULL terminated string in format <dirName/Filename>

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX00900A (XX – see result code).	Result Code

Result Code

Value	Code	Description
0x00	OK	Command completed successfully.
0x01	BIOS_HECI_STATUS_INVALID_PARAM	
0x02	BIOS_HECI_STATUS_FILE_NOT_FOUND	
0x03	BIOS_HECI_STATUS_AFTER_EOS	
0x04	BIOS_HECI_STATUS_DIR_LOCKED	
0x05	BIOS_HECI_STATUS_NVM_INACCESSIBLE	
0x06	BIOS_HECI_STATUS_TOO_MANY_DIR	
0x07	BIOS_HECI_STATUS_COUNTER_INVALID	
0x08	BIOS_HECI_STATUS_SIGN_INVALID	
0x09	BIOS_HECI_STATUS_POST_EOP	
0x0A	BIOS_HECI_STATUS_POST_CORE_BIOS_DONE	
0x0B	BIOS_HECI_STATUS_NO_QUOTA	
0x0F	BIOS_HECI_STATUS_ERROR	
0xFF	GEN_ERROR	General error with command.

Details

Lock BIOS dir from writes.

28.7.4.7 End of Services

Availability

	Before DID	Post DID	Post CBD	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	Yes	Not locked by CBD.
Post Manufacturing	No	Yes	Yes	Yes	Yes	
Recovery	No	Yes	Yes	Yes	Yes	

**Handler**

CMO Process	BUP Process	MBP Entry
MCA	Yes	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x0000011A	MKHI Group ID GROUP ID 0x0A Command 0x11

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX009110A (XX – see result code).	Result Code

Result Code

Value	Code	Description
0x00	OK	Command completed successfully.
0x01	BIOS_HECI_STATUS_INVALID_PARAM	
0x02	BIOS_HECI_STATUS_FILE_NOT_FOUND	
0x03	BIOS_HECI_STATUS_AFTER_EOS	
0x04	BIOS_HECI_STATUS_DIR_LOCKED	
0x05	BIOS_HECI_STATUS_NVM_INACCESSIBLE	
0x06	BIOS_HECI_STATUS_TOO_MANY_DIR	
0x07	BIOS_HECI_STATUS_COUNTER_INVALID	
0x08	BIOS_HECI_STATUS_SIGN_INVALID	
0x09	BIOS_HECI_STATUS_POST_EOP	
0x0A	BIOS_HECI_STATUS_POST_CORE_BIOS_DONE	
0x0B	BIOS_HECI_STATUS_NO_QUOTA	
0x0F	BIOS_HECI_STATUS_ERROR	
0xFF	GEN_ERROR	General error with command.

Details

Stop requests for NVM offload over MKHI.

28.7.4.8 CURRENT BOOT MEDIA

Availability

	Before DID	Post DID	Post CBD	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	No	Yes	Not locked by CBD.
Post Manufacturing	No	Yes	Yes	No	Yes	
Recovery	No	Yes	Yes	Yes	Yes	

Handler

CMO Process	BUP Process	MBP Entry
MCA	Yes	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00000120A	MKHI Group ID GROUP ID 0xA Command 0x12

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX00920A (XX – see result code).	Result Code
4	BPDTOffset	Various	Offset from start physical partition mentioned in physical data.
4	Physical Data	0000b: emmc 0001b: UFS 0010b: SPI	BIT[3:0] Boot Device
4	Physical Device Area	0000b: Boot1 (eMMC/UFS) 0001b: Boot2 (eMMC/UFS) 0000b: Host Region (SPI)	BIT[3:0] = Device Area Enum based on Physical Data.
4	Logical Data	00b: BPDT1 Selected	BIT[1:0] – Logical Partition.



Size Bytes	Field Name	Field Values	Description
		01b: BPDT2 Selected	

Result Code

Value	Code	Description
0x00	OK	Command completed successfully.
0x01	BIOS_HECI_STATUS_INVALID_PARAM	
0x02	BIOS_HECI_STATUS_FILE_NOT_FOUND	
0x03	BIOS_HECI_STATUS_AFTER_EOS	
0x04	BIOS_HECI_STATUS_DIR_LOCKED	
0x05	BIOS_HECI_STATUS_NVM_INACCESSIBLE	
0x06	BIOS_HECI_STATUS_TOO_MANY_DIR	
0x07	BIOS_HECI_STATUS_COUNTER_INVALID	
0x08	BIOS_HECI_STATUS_SIGN_INVALID	
0x09	BIOS_HECI_STATUS_POST_EOP	
0x0A	BIOS_HECI_STATUS_POST_CORE_BIOS_DONE	
0x0B	BIOS_HECI_STATUS_NO_QUOTA	
0x0F	BIOS_HECI_STATUS_ERROR	
0xFF	GEN_ERROR	General error with command.

Details

1. BUP storage component reads AON SRAM "Image Source Info" struct:
 - a. Copies Preferred Available boot source Byte[0] bits [3:1] to Payload Physical Data bits [2:0].
 - b. Copies Preferred Available boot source Byte[2] bits [7:6] to Payload Logical Data bits [1:0].
 - c. Sets Payload Physical Data bits [7:4] according to steps #1a and #1b.
 - d. Populates Bytes [3:0] according to steps #1a-#1c (in most cases it would be 0, except for case of BPDT2 on SPI Host region).
2. BUP storage component populates MBP with above payload and publishes it in snowball.
3. When MCA component gets loaded, it takes above payload from snowball, and returns it whenever the MKHI command is called.

28.7.5 MKHI – Secure Boot Group

```
#define MKHI_SECURE_BOOT_GROUP_ID 0xC
```

28.7.5.1 VERIFY MANIFEST

Availability

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	
Post Manufacturing	No	Yes	Yes	Yes	
Recovery	No	No	Yes	Yes	

Processes

CMO Process	BUP Process	MBP Entry
Policy	Yes	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x0000010C	MKHI Group ID GROUP ID 0xC Command 0x1
4	UsageIndex		The usage index.
4	ManifestSize	0 – 2000	Size in DWORDS of manifest + ext. Max is 2K DWORDs or 8KB
4	MetaContentSize	0 – 8000	Size in bytes of metadata extension.
1	Flags	00000001'b – Manifest Data 00000010'b – Metadata Data 00000100'b – Start over 00001000'b – Last Chunk	01'b – Chunk has manifest data in it. 10'b Chunk has metadata data in it. 100'b Start over verify. 1000'b Last chunk. Do verify. Note: Bit0 and Bit1 are mutually exclusive. Bit2 and Bit3 are mutually exclusive. Bit0 or BIT1 can be set with BIT2 or BIT3.
2	ChunkSize	1 - 484	Size of chunk in Data
1	Reserved	0x0	Reserved
< 488	Data	Manifest or Metadata Data buffer	The chunk of data.

**MKHI Response**

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX00810c (XX – see result code).	Result Code
1	Flags	See request	The flags on request
2	ChunkSize	See request	The chunk size processed
1	Reserved	0x0	Reserved

Result

Value	Code	Description
0x00	OK	Manifest successfully verified
0x01	OK_CHUNK	Successful chunk process
0x05	SIZE_ERROR	Command size does not match expected value.
0x0B	EMANHDR	Manifest Header Invalid
0x0C	EINVSIG	Manifest sig invalid
0x8D	STATUS_INVALID_COMMAND	Command not supported.
0xFE	RETRY	Not CMO T.C. retry command.
0xFF	GEN_ERROR	General error with manifest verify.

Details

- Host – Host sends the data in chunks.
 - Manifest data is the manifest header, the manifest and the manifest extension.
 - Metadata data.
 - Host will send first chunk with flags bit2 set to start command.
 - Host will send last chunk with flags bit3 set to end command and get verification results.
 - Host will retry upon receiving RETRY(0xFE) until CMO T.C. is received.
- BUP
 - BUP will allocate buffers off heap equal to the manifest size and the metadata size in the command.
 - BUP will free buffer on the command end, when a new start over bit is detected, when BUP no longer polls HECI.
 - BUP will implement `ldr_verify_manifest_for_usage` API directly.
- Policy
 - Policy will allocate buffers off heap equal to the manifest size and the metadata size in the command.
 - Policy will free the buffer on the command end, when new start over bit is detected or when the connection is closed or HECI link reset.
 - Policy will call loader API directly.

28.7.5.2 GET ARB STATUS

Availability

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	
Post Manufacturing	No	Yes	Yes	Yes	
Recovery	No	No	No	No	Not available.

Handler

CMO Process	BUP Process	MBP Entry
Policy	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x0000020C	MKHI Group ID GROUP ID 0xC Command 0x2

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX00820c (XX – see result code).	Result Code
16	Dirty SVNs	0 (bit) – Not Pending Update 1 (bit) – Pending Update.	Bitmap indicates if ARB is pending update per component. unsigned int dirty_svns[32] bit0 – component 0 bit127 – component 128
512	Cur SVNs	SVN Number	The current SVN for each component. 4 bytes per component. unsigned int cur_svns[128] entry 0 – component 0 entry 127 – component 128
512	New SVNs	SVN Number	The pending SVN for each component. 4 bytes per component.



Size Bytes	Field Name	Field Values	Description
			unsigned int new_svns[128] entry 0 – component 0 entry 127 – component 128
16	ARBs Enabled	0 (bit) – Disabled. 1 (bit) – Enabled.	Bitmap indicates if ARB is enabled per component. unsigned int enabled_svns[32] bit0 – component 0 bit127 – component 128
16	Dynamic ARB Status	0 (bit) – Not performed. 1 (bit) – Performed.	Bitmap indicates if manifest was verified by CSE FW loader. unsigned int dynamic_svns[32] bit0 – component 0 bit127 – component 128

Result

Value	Code	Description
0x00	OK	SVN info populated correctly.
0x22	ENINVAL	Error populating SVNs.
0x05	SIZE_ERROR	Command size does not match expected value.
0x89	STATUS_NOT_SUPPORTED	After EOP
0x8D	STATUS_INVALID_COMMAND	Command not supported.
0x99	EGERERIC	Error populating SVNs.
0xFE	RETRY	Not CMO T.C. retry command.

Details

Host – Calls API on boot after ifwi update.

Policy – Calls loader API ldr_get_anti_roll_back_status() and returns SVN info to host.

28.7.5.3 COMMIT ARB SVN UPDATES

Availability

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	
Post Manufacturing	No	Yes	Yes	Yes	

	Before DID	Post DID	Post EOP	Before FID	Notes
Recovery	No	No	No	No	Not available.

Handler

CMO Process	BUP Process	MBP Entry
Policy	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x0000030C	MKHI Group ID GROUP ID 0xC Command 0x3
16	SVNs to Commit	0 (bit) – Don't Commit 1 (bit) – Commit	Bitmap indicates which components to commit ARB. unsigned int commit_svns[32] bit0 – component 0 bit127 – component 128

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX00820c (XX – see result code).	Result Code

Result

Value	Code	Description
0x00	OK	SVN info populated correctly.
0x22	ENINVAL	Error populating SVNs.
0x05	SIZE_ERROR	Command size does not match expected value.
0x89	STATUS_NOT_SUPPORTED	After EOP
0x8D	STATUS_INVALID_COMMAND	Command not supported.
0x99	EGENERIC	Error populating SVNs.
0xFE	RETRY	Not CMO T.C. retry command.

Details

Host – Call API after IFWI update to commit SVNs.



Policy – Call Loader API Idr_commit_anti_roll_back_updates() to perform update.

28.7.6 MKHI – DNX Group

```
#define MKHI_DNX_GROUP_ID 0x0D
```

28.7.6.1 CSE DNX REQ Set

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	No	Yes	
Post Manufacturing	No	Yes	No	Yes	
Recovery	No	No	No	No	Not available.

Handler

CMO Process	BUP Process	MBP Entry
Policy	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x0000010D	MKHI Group ID GROUP ID 0xD Command 0x1

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX00820c (XX – see result code).	Result Code
4	ReqBiosAction	0 – Continue Post 1- Reset that power cycles both CSE and host	BXT – Cold reset.

Result

Value	Code	Description
0x00	OK	DNX breadcrumb set.
0x05	SIZE_ERROR	Command size does not match expected value.
0x89	STATUS_NOT_SUPPORTED	After EOP



Value	Code	Description
0x8D	STATUS_INVALID_COMMAND	Command not supported.
0x99	EGERERIC	Error setting DNX breadcrumb.

Details

Host – Call API to set CSE DNX REQ.

- MBP IAFW DNX REQUEST entry bit is set indicating OS DNX was requested.
- IBB detects OBB is corrupt and wants CSE to enter DNX.
- Other host entered reasons from IBB.

Policy will set Bit6 RBE_DNX_REQ in DNX breadcrumb which will indicate to RBE to enter DNX.

DNX Tool will clear the CSE DNX REQ.

The following boundary condition exists:

- CSE FW is missing beyond BUP
- Iafw is missing beyond IBB.
- In this case IBB would send this message to CSE, but Policy Module does not exist therefore this message stays stuck in HECI CB. The only way to recover from this situation and make the platform enter DNX is via DNX HW strap. Deliberate decision to reduce complexity.

28.7.6.2 IAFW DNX REQ Clear

Availability

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	No	Yes	
Post Manufacturing	No	Yes	No	Yes	
Recovery	No	No	No	No	Not available.

Handler

CMO Process	BUP Process	MBP Entry
Policy	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x0000020D	MKHI Group ID



Size Bytes	Field Name	Field Values	Description
			GROUP ID 0xD Command 0x2

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX00820D (XX – see result code).	Result Code
4	Flags	00b: Reset DnX protocol (no CSE /device reset) by cancelling currently active command (if any) and wait for the next command 01b: Perform Deep reset of the SoC after sending response. After deep reset, the security policy (SoC unlock state) must be reset to default 10b: Perform Shallow reset after sending response 11b: Perform Deep reset of the SoC after sending response. After deep reset, the current security policy (SoC unlock state) must be retained	Note: See DNX Support FAS Bit[1:0] – Flags Bit[31:2] – Reserved and set to zero.

Result

Value	Code	Description
0x00	OK	Success Device lifecycle token must present
0x05	SIZE_ERROR	Command size does not match expected value.
0x89	STATUS_NOT_SUPPORTED	Availability Error: Command was sent incorrectly, due to: Command sent prior to DID. Command sent after EOP/Post EOM
0x8D	STATUS_INVALID_COMMAND	Command not supported.
0xFF	STATUS_UNDEFINED	Undefined Error (not expected, BIOS may continue with update)

Details

Host – Sends to clear the OS DNX request.

Policy – Clears DNX breadcrumb for OS DNX Request.

CSE is initiating the reset.

28.7.7 MKHI – IFWI Update

```
#define MKHI_IFWI_UPDATE_GROUP_ID 0x20
```

28.7.7.1 IFWI PREPARE FOR UPDATE COMMAND

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	Requires UMA. Before FID returns retry.
Post Manufacturing	No	Yes	No	Yes	
Recovery	No	Yes	No/Yes	Yes	BUP checks Manuf mode for EOP handling.

Handler

CMO Process	BUP Process	MBP Entry
Policy	Yes	None

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00000120	GROUP ID 0x20 Command 0x01
1	Reset type	0x00-0x01	<p>Type of reset needed after completion of FW update. 0x00 - Natural operation to clear PCH/SoC context. 0x01 - Force Type 7 GRST</p> <p>0x00 should be used in most cases, as it will ensure that all PCH/SoC context is cleared on next platform reset. CSE will perform the relevant operation based on platform behavior. For APL, this operation will translate to Unconditional Power Down.</p> <p>0x01 should be used specifically in case type 7 GRST is desired. Most context will be cleared and the platform will boot to S0 after the reset. This value might be useful in scenarios of manufacturing where platform boot to S0 is more important than clearing 100% of the context.</p>



MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX008120 (XX – see result code).	Result Code
1	Flags	0x00	<p>01b – Try again. FW initialization not complete and/or IFWI prepare for update is still in process. Try again. Continue to poll.</p> <p>00b – Reserved 10b – IFWI prepare for update is successful. Host/BIOS to stop polling Reset to BUP, blocked NVM access, running from non-NVM. 11b – Error (See error code below)</p> <p>Host polls on flags field until 10b indicates CSE FW has stopped NVM access and switched to run from Non-NVM case.</p> <p>CSE will continue to run from Non-NVM until host issues GRST.</p>

Result

Value	Code	Description
0x00	OK	Success
0x05	SIZE_ERROR	Command size does not match expected value.
0x89	STATUS_NOT_SUPPORTED	Availability Error: Command was sent incorrectly, due to: Command sent prior to DID. Command sent after EOP/Post EOM
0x8D	STATUS_INVALID_COMMAND	Command not supported.
0xFC	STATUS_PROHIBITED	NOT RELEVANT FOR BXT BUT MIGHT BE RELEVANT IN FUTURE. CSE is in some special security sensitive state in which it does not allow IFWI update to occur.



Value	Code	Description
0xFE	STATUS_AUDIT_LOG_ERROR	System Audit Log Error, BIOS should halt the FW Update flow and notify that operation is denied since AMT Auditing does not allow this operation (NOT for BXT)
0xFF	STATUS_UNDEFINED	Undefined Error (not expected, BIOS may continue with update)

Details

Host – Call API to prepare CSE for IFWI prepare for update. Host does the following:

- Sends IFWI Prepare For Update Command MKHI request.
- Receives IFWI Prepare For Update Command MKHI response.
- Check flag indicates 01b to indicate CSE is ready to begin update.
 - If flag is set to 00b repeat from first step.
 - If flag is set to 01b then CSE reset will occur and poll for CSE_RST bit.
 - If result in MKHI result indicates failure then error in ifwi update flow and exit.

Note: This command is expected to generate a CSE reset therefore host must follow exactly flow outlined in HECI definitions specifically section **Host Sending Message to ME Flow** and **ME Sending Message to Host Flow**. Specifically on receiving response to IFWI prepare for update if reading CSE_RDY is cleared then CSE has undergone a reset and host should wait for CSE_RST bit to be set before executing flow in **ME Initiating the reset of the HECI**.

Note: If the MKHI response is successfully read and flag is set to 01b then CSE_RST will be set after CSE partition reset.

If flag is set to 10b then CSE has blocked NVM access and is running from UMA.

Policy – Upon receiving the command policy will call maestro to perform a CSE reset and send response. Upon error the corresponding flags/result will be set and policy will not call maestro to initiate reset.

28.7.7.2 DATA CLEAR COMMAND

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	Only in recovery.
Post Manufacturing	No	Yes	No	Yes	
Recovery	No	Yes	No/Yes	Yes	BUP checks Manuf mode for EOP handling.

Handler

CMO Process	BUP Process	MBP Entry
Policy	Yes	

**MKHI Request**

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00000220	GROUP ID 0x20 Command 0x02

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX008220 (XX – see result code).	Result Code

Result

Value	Code	Description
0x00	OK	Success Device lifecycle token must present
0x01	STATUS_DATA_CLEAR_LOCKED	Command sent AFTER DATA CLEAR LOCK MEI command
0x02	STATUS_DATA_CLEAR_NVAR_FAILURE	DATA CLEAR NVAR set to not allow DATA CLEAR
0x05	SIZE_ERROR	Command size does not match expected value.
0x89	STATUS_NOT_SUPPORTED	Availability Error: Command was sent incorrectly, due to: Command sent prior to DID. Command sent after EOP/Post EOM
0x8D	STATUS_INVALID_COMMAND	Command not supported.
0xFB	STATUS_IFWI_PREPARE_FOR_UPDATE_REQUIRED	IFWI Prepare for update needs to be successful before Data clear will be performed.
0xFC	STATUS_PROHIBITED	CSE is in some special security sensitive state in which it does not allow IFWI update to occur.
0xFE	STATUS_AUDIT_LOG_ERROR	System Audit Log Error, BIOS should halt the FW Update flow and notify that operation is denied since AMT Auditing does not allow this operation



Value	Code	Description
0xFF	STATUS_UNDEFINED	Undefined Error (not expected, BIOS may continue with update)

Details

NOTES:

1. This table only describes when it COULD be successful, key contributing factor is after a successful IFWI PREPARE FOR UPDATE has been executed AND if NVAR has been configured to allow for it and if DATA clear lock has not been sent.
2. If Flash descriptor override pin is asserted (FDO) then Data clear command will be allowed regardless of data clear lock API and/or NVAR to disable data clear.

Data Clear command replacing HMRFPO for specific FW data clear steps.

The Data Clear command will be accepted only in one of the following conditions:

The command will be accepted after successful IFWI prepare for update command.

28.7.7.3 DATA CLEAR LOCK

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	Data clear is allowed pre-EOM.
Post Manufacturing	No	Yes	No	Yes	
Recovery	No	Yes	No	Yes	

Handler

CMO Process	BUP Process	MBP Entry
Policy	Yes	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00000420	GROUP ID 0x20 Command 0x04

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX008420 (XX – see result code).	Result Code

**Result**

Value	Code	Description
0x00	OK	Success command accepted.
0x05	SIZE_ERROR	Command size does not match expected value.
0x89	STATUS_NOT_SUPPORTED	Sent after EOP.
0x8D	STATUS_INVALID_COMMAND	Command not supported.
0xFF	STATUS_UNDEFINED	Undefined Error (not expected, BIOS may continue with update)

Details

Note: If Flash descriptor override pin is asserted (FDO) then Data clear command will be accepted, but it has no affect on gating data clear command.

There are two ways of locking/preventing DATA Clear.

1. DATA CLEAR LOCK API which sets internal state to stop data clear from happening until next platform reset. CSE FW should maintain lock across S3 resume due to the BIOS not sending EOP or data clear messages.
2. DATA Clear LOCK NVAR – Set by FIT which prevent DATA CLEAR from ever happening on that platform POST EOM.

DATA CLEAR LOCK API is called before EOP and will disable the “DATA CLEAR” MEI command until next platform reset. DATA CLEAR LOCK API will change the internal state to prevent DATA clear. This state will not have effect DATA CLEAR pre-EOM or FDO.

DATA CLEAR LOCK NVAR has no effect if pre-EOM or FDO is asserted and BUP reaches recovery.

Note: DATA Clear will always work pre-EOM or FDO. DATA CLEAR LOCK NVAR or DATA CLEAR LOCK API will not prevent DATA CLEAR pre-EOM.

Note: If “DATA CLEAR” LOCK NVAR is set to disable by OEM in FIT and EOM is set, refurbishing the device will require physical access to pull the FDO strap for unlocking CSE data region on SPI or use DnX for data clear of CSE data on eMMC/UFS.

Note: Intel strongly recommends that Data clear lock is called before any OROMs or untrusted/authenticated IAFW is run on the platform. After the Data Clear lock is any “DATA CLEAR” MEI command will fail until next platform reset.

Note: This is advised for OEMs who are not supporting UEFI secure boot and secure update (or other means to secure their BIOS).

28.7.7.4 UPDATE IMAGE CHECK

Implementation note for CSE: CSE must use RSO for access to the memory addresses provided. Otherwise, CSE might accidentally access its own device space, and this can be used for attacks.



	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	No	Yes	
Post Manufacturing	No	Yes	No	Yes	
Recovery	No	Yes	No	Yes	

Handler

CMO Process	BUP Process	MBP Entry
Policy	Yes	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00000620	GROUP ID 0x20 Command 0x06
4	Image Base Addr – lower 32 bits		Lower 32-bits of the host memory address containing the image which CSE should inspect. NOTE: Should be 4 K aligned
4	Image Base Addr – upper 32 bits		Upper 32-bits of the host memory address containing the image which CSE should inspect. NOTE: Entire IFWI should not be larger than 32 MBytes
4	Image size		Size of the image to inspect.

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX008620 (XX – see result code).	Result Code
4	Image Error Value	0x00	Only has meaning in case Result Code indicated IMAGE_FAILED. 0x7FF_FFFF: Undefined error. This data structure is references from ROM 13.3 Critical Portion Image Error Values. Using this structure in case in the future we want to add more checks to this API.



Size Bytes	Field Name	Field Values	Description
			<p>These bits only have meaning if an error has been found.</p> <ul style="list-style-type: none">o Bit [31] == 0<input type="checkbox"/> Bits [30:28] == 101b: Version (Secure / Unsecure) – stands for SVN downgrade or VCN downgrade for all components including CSE- this field will be set if SVN or VCN of a component in the update image is less than the SVN or VCN of that component that is already loaded. VCN by definition are not backwards compatible. If any component's SVN/VCN in the update image is greater than what is currently loaded. then the flag Rollback possible must be set to '01' – rollback not possible.Bits [30:28] == 011b: Image Invalid : If the image cannot be correctly parsed, this error will be returned.Bits [30:28] == Other Values: Undefined.Bits [27:25] == 011b: DRAM.Bits [24:21] == 0001b: Host accessible memory for Update image checkBit [20] == 1b. Logical Partition Error.Bit [19] == 0b: Boot Logical Partition Error.Bits [18:16] == 101b: Manifest.Bits [15:0]:<ul style="list-style-type: none">o 0b = Non CSE component SVN Downgrade (see ARB SVN Downgrade Bitmap)o 1b = CSE SVN Downgrade

Size Bytes	Field Name	Field Values	Description
			<ul style="list-style-type: none"> o '10b' = CSE VCN downgrade – CSE is being downgraded and will not be compatible o All others reserved
16	ARB SVN downgrade bitmap.	0 (bit) – SVN equal or greater than current. (No Error) 1 (bit) – SVN Downgrade from current	Bitmap indicates if downgrading SVN per component. bit0 – component 0 bit127 – component 128
1	Roll back possible		Only possible if status was successful <ul style="list-style-type: none"> - possible - not possible <p>[This condition is only possible if CSE's VCN or SVN is greater than what is currently running in the platform]</p>

Result

Value	Code	Description
0x00	OK	Success. Image was checked (best effort) and verified to be appropriate for FW update.
0x1	IMAGE_FAILED	Failure. If this image is FLASHed, some error will occur. Details of that error are detailed in "Image Error Value" field in the response.
0x02	IMG_SIZE_INVALID	Image size parameter which was provided is invalid, e.g: 0. Not 4K aligned. Too large.
0x05	SIZE_ERROR	Command size does not match expected value.
0x89	STATUS_NOT_SUPPORTED	Availability Error: Command was sent before IFWI Prepare For Update returned SUCCESS.
0x8D	STATUS_INVALID_COMMAND	Command not supported.
0xFF	STATUS_UNDEFINED	Undefined Error (not expected, BIOS may continue with update)

Update image check implementation notes:



This is in place to check the SVN of the incoming image against current image.

Host will pass in a pointer to where the update image is in memory. BUP loader will perform verification checks and provide the appropriate error to host.

All checks and authentication should be performed against information in the internal CSE SRAM. This is to prevent host/BIOS attacker from manipulating the memory during the check. The intention protection is to protect CSE assets from attacks such as buffer overflow and heap attacks. BIOS/Host could still manipulate the image in DRAM or just flash a malware IFWI image.

Update Image check API shall perform checks/authentication:

If OEM Public Key hash == 0x0s

Will not look for OEM KM and not provide any authentication services for OEM signed partitions

Else If OEM Public Key hash != 0x0s

Will look for OEM KM and use the OEM KM to authenticate non-Intel sub partitions.

BUP loader shall parse through update IFWI image and authenticate each CSE sub-partition. (Example RBE, BUP, Main)

BUP Loader must verify SVNs of all non-CSE IFWI components in the update image compared against what is in the ARB SVN file. If anti-rollback is disabled, then skip this check for non-CSE IFWI components.

For all CSE components check against the VCN and SVN of the SPIE (Signed Package Info Extension) Header of the RBE. CSE VCN of the RBE of the update image checked against what is loaded in the current RBE manifest. VCN check should be against both copies of RBE (BPDT1 and 2 if it exists in both places)

If new image CSE's SVN / VCN > current one, BUP must set "rollback not possible" flag in the response.

Note: This is not available after POST due to concerns for having DMA from host to CSE after EOP as well as having a single unified flow.

To address security concerns there will be additional checks to reduce the attack surface against CSE.

What this provides:

1. Mechanism to notify if IFWI components' SVNs are being downgraded
2. Mechanism to notify BIOS that a rollback is possible after update (will fail if CSE SVN or VCN has been incremented).
3. Mechanism to notify BIOS that of CSE VCN is being downgraded
4. Authentication for CSE IFWI sub-partitions



5. Authentication for OEM signed IFWI sub-partitions (if OEM KM hash has been burned)

What this does NOT provide

1. This is ***NOT*** a comprehensive check that will be done when CSE loads modules. This is a quick check on SVN/VCN and to note if a rollback would be possible on CSE side.
 - a. Any checks specific to component will note be included (example PMC, ISH)
 - b. Build version differ on CSE components would not be checked

Note: This API check does not substitute for an Overall IFWI authentication. The overall authentication is required to ensure this is a valid OEM update.

28.7.8 MKHI – General Group

```
#define MKHI_GEN_GROUP_ID 0xFF
```

The following group is the general MKHI message group. This group contains general MKHI commands and is handled by Policy Manager in CM0.

28.7.8.1 MKHI VERSION COMMAND

Availability

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	
Post Manufacturing	No	Yes	Yes	Yes	
Recovery	No	No	No	No	Not available.

Handler

CMO Process	BUP Process	MBP Entry
Policy	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x000001FF	MKHI Group ID GROUP ID 0xFF Command 0x1

**MKHI Response**

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX0081FF (XX – see result code).	Result Code
2	Minor	0x1	Minor Version
2	Major	0x1	Major Version

Result

Value	Code	Description
0x00	OK	MKHI Version Returned.
0x05	SIZE_ERROR	Command size does not match expected value.
0x8D	STATUS_INVALID_COMMAND	Command not supported.

Details

Policy returns the MKHI version.

28.7.8.2 GET FW VERSION COMMAND

Availability

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	No	Yes	
Post Manufacturing	No	Yes	No	Yes	
Recovery	No	No	No	No	Not available.

Handler

CMO Process	BUP Process	MBP Entry
Policy	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x000002FF	MKHI Group ID GROUP ID 0xFF Command 0x2

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX0082FF (XX – see result code).	Result Code
2	CodeMinor	0 – 0xFF	The minor version
2	CodeMajor	0 – 0xFF	The major version.
2	CodeBuildNo	0 – 0xFF	The code build number
2	CodeHotFix	0 – 0xFF	The code hotfix
2	NFTPMinor	0 – 0xFF	The NFTP Minor version.
2	NFTPMajor	0 – 0xFF	The NFTP Major version.
2	NFTPBuildNo	0 – 0xFF	The NFPT build number
2	NFTPHotFix	0 – 0xFF	The NFPT hot fix number.
2	FITCMinor	0 – 0xFF	The FITC minor version.
2	FITCMajor	0 – 0xFF	The FITC major version.
2	FITCBuildNo	0 – 0xFF	FITC build number.
2	FITCHotFix	0 – 0xFF	The FITC hot fix.

Result

Value	Code	Description
0x00	OK	FW Version Returned.
0x05	SIZE_ERROR	Command size does not match expected value.
0x8D	STATUS_INVALID_COMMAND	Command not supported.

Details

Policy will retrieve the version numbers from the manifest and populate the response.

28.7.8.3 GET IMAGE FW VERSIONS

Availability

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	
Post Manufacturing	No	Yes	Yes	Yes	
Recovery	No	No	No	No	Not available.

**Handler**

CMO Process	BUP Process	MBP Entry
Policy	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x00001CFF	MKHI Group ID GROUP ID 0xFF Command 0x1C

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX009CFF (XX – see result code).	Result Code
4	NumModules		The number of modules retrieved.
NumModules X 20 Bytes	ModuleEntries		Array of versions. Size is determined by NumModules times the size of each version entry. See def below.

ModuleEntries

Size Bytes	Field Name	Field Values	Description
12	EntryName		ASCII string.
2	Major		Major Number
2	Minor		Minor Number
2	HotFix		Hotfix Number
2	Build		Build number.

Result

Value	Code	Description
0x00	OK	FW Version Returned.
0x05	SIZE_ERROR	Command size does not match expected value.
0x8D	STATUS_INVALID_COMMAND	Command not supported.

Details

Policy will return FW versions for each binary in the IFWI which has CSS manifest.

The list below is what we expect in BXT based solutions.

- CSE
- Iunit
- ISH
- BIOS
- PMC
- SMIP
- IDLM

28.7.9 MKHI – BUP Common Group

```
#define MKHI_GROUP_ID_BUP_COMMON 0xF0
```

28.7.9.1 DRAM INIT DONE Command

Availability

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	No	Yes	
Post Manufacturing	No	Yes	No	Yes	
Recovery	No	Yes	No	Yes	

Handler

CMO Process	BUP Process	MBP Entry
Policy	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x000001F0	MKHI Group ID GROUP ID 0xF0 Command 0x01
8	BiosImrDisable		Bitmask of disabled IMRs



Size Bytes	Field Name	Field Values	Description
4	BiosMinImrsBa		Min base address of BIOS IMR sizes
1	Flags		Bit[0] – FfsEntryExit Bit[1] – NonDestructiveAliasCheck Bit[31:2] – Reserved
1	MemStatus	0 - BIOS_MSG DID_SUCCESS 0x1 - BIOS_MSG DID_NO_MEMORY 0x2 - BIOS_MSG DID_INIT_ERROR 0x3 - BIOS_MSG DID_MEM_NOT_PERSERVED	Memory init status

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX0081F0 (XX – see result code).	Result Code
4	ImrsSortedRegionBa		BA of sorted IMR
4	ImrsSortedRegionLen		Len of sorted IMR
1	OemSettingsRejected		OEM IMRS rejected
3	Reserved		
1	BiosAction	1 - DID_ACK_NON_PCR 2 - DID_ACK_PCR 3 - DID_ACK_RSVD3 4 - DID_ACK_RSVD4 5 - DID_ACK_RSVD5 6 - DID_ACK_GRST 7 - DID_ACK_CONTINUE_POST	Action for BIOS to take
3	Reserved		

Result

Value	Code	Description
0x00	OK	DRAM INIT DONE message accepted.
0x05	SIZE_ERROR	Command size does not match expected value.
0x8D	STATUS_INVALID_COMMAND	Command not supported.

Details

28.7.9.2 ME BIOS PAYLOAD

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	No	Yes	No	Yes	
Post Manufacturing	No	Yes	No	Yes	
Recovery	No	Yes	No	Yes	

Handler

CMO Process	BUP Process	MBP Entry
Policy	No	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x000002F0	MKHI Group ID GROUP ID 0xF0 Command 0x02
4	Flags	0b – MBP requested 1b – MBP Skip	BIT0 – If set means do not send MBP.

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX0081F0 (XX – see result code).	Result Code
4	MBP Header	See MBP Header	The MBP header.
4 - 504	MBP Item Data	See MBP Item Def	The MBP items. See the MBP item defs.

Result

Value	Code	Description
0x00	OK	ME BIOS PAYLOAD returned.
0x05	SIZE_ERROR	Command size does not match expected value.
0x8D	STATUS_INVALID_COMMAND	Command not supported.



28.7.9.3 CSE ENABLE COMMAND

Availability

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	Yes	No	No	Yes	
Post Manufacturing	Yes	No	No	Yes	
Recovery	No	No	No	No	Not available.

Handler

CMO Process	BUP Process	MBP Entry
None – BUP only	Yes	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x000003FO	MKHI Group ID GROUP ID 0xF0 Command 0x03

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX0083FO (XX – see result code).	Result Code

Result

Value	Code	Description
0x00	OK	CSE Enable message accepted.
0x05	SIZE_ERROR	Command size does not match expected value.
0x8D	STATUS_INVALID_COMMAND	Command not supported.

28.7.9.4 Clear MANUF OVERRIDE COMMAND

Availability

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	Yes	No	No	Yes	
Post Manufacturing	Yes	No	No	Yes	

	Before DID	Post DID	Post EOP	Before FID	Notes
Recovery	No	No	No	No	Not available.

Handler

CMO Process	BUP Process	MBP Entry
None – BUP only.	BUP	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x000004F0	MKHI Group ID GROUP ID 0xF0 Command 0x04

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0XX0084F0 (XX – see result code).	Result Code

Result

Value	Code	Description
0x00	OK	Manuf override clear accepted.
0x05	SIZE_ERROR	Command size does not match expected value.
0x8D	STATUS_INVALID_COMMAND	Command not supported.

28.7.9.5 TRACING ENABLED Command**Availability**

	Before DID	Post DID	Post EOP	Before FID	Notes
Manufacturing	Yes	No	No	Yes	
Post Manufacturing	Yes	No	No	Yes	
Recovery	No	No	No	No	Not available.

**Handler**

CMO Process	BUP Process	MBP Entry
None – BUP only.	Yes	

MKHI Request

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0x000005F0	MKHI Group ID GROUP ID 0xF0 Command 0x05

MKHI Response

Size Bytes	Field Name	Field Values	Description
4	MKHI Header	0xXX0081F0 (XX – see result code).	Result Code

Result

Value	Code	Description
0x00	OK	Tracing enabled message accepted.
0x05	SIZE_ERROR	Command size does not match expected value.
0x8D	STATUS_INVALID_COMMAND	Command not supported.

28.8 HECI2 NVM Offload

HECI2 is used by BIOS SMM to access NVM.

28.8.1 READ DATA

Availability

	Before DID	Post DID	Before EOP	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	Yes	
Post Manufacturing	No	Yes	Yes	Yes	Yes	
Recovery	No	No	No	No	No	Not available.

READ DATA Request

Size Bytes	Field Name	Field Values	Description
32	HMAC-SHA256 Signature	HMAC-SHA256	The signature of entire data including counter but minus this field. Note: Place here to keep hashed fields contiguous for programmer ease.
1	Req_resp[0] Cmd_id[7]	0x02	Req_resp[0] 0'b – BIOS_TO_CSE Cmd_id[7] 0x1 – READ_DATA
4	Counter	0 – 0xFFFFFFFF	The monotonic value from the host. CSE will confirm this is larger than the last MC value starting with the init value.
25	FilePath	ASCII	NULL terminated string in format <dirName/Filename>
4	Offset		Offset within the file.
2	DataSize		Number of bytes to Read.

READ DATA Response

Size Bytes	Field Name	Field Values	Description
32	HMAC-SHA256 Signature	HMAC-SHA256	The signature of entire data including counter but minus this field. Note: Place here to keep hashed fields contiguous for programmer ease.
1	Req_resp[0] Cmd_id[7]	0x03	Req_resp[0] 1'b – CSE_TO_BIOS Cmd_id[7] 0x1 – READ_DATA
4	Counter	0 – 0xFFFFFFFF	CSE will return the request counter value.



Size Bytes	Field Name	Field Values	Description
1	Status	0x0 – 0xF	BIOS_HECI_STATUS_OK 0x0 BIOS_HECI_STATUS_INVALID_PARAM 0x1 BIOS_HECI_STATUS_FILE_NOT_FOUND 0x2 BIOS_HECI_STATUS_AFTER_EOS 0x3 BIOS_HECI_STATUS_DIR_LOCKED 0x4 BIOS_HECI_STATUS_NVM_INACCESSIBLE 0x5 BIOS_HECI_STATUS_TOO_MANY_DIR 0x6 BIOS_HECI_STATUS_COUNTER_INVALID 0x7 BIOS_HECI_STATUS_SIGN_INVALID 0x8 BIOS_HECI_STATUS_POST_EOP 0x9 BIOS_HECI_STATUS_POST_CORE_BIOS_DONE 0xA BIOS_HECI_STATUS_NO_QUOTA 0xB BIOS_HECI_STATUS_ERROR 0xF
2	DataSize		Size of data following
DataSize	Data		The data for the read file.

28.8.2 WRITE DATA

Availability

	Before DID	Post DID	Before EOP	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	Yes	On shared NVM HECI2 post EOS requires storage proxy.

	Before DID	Post DID	Before EOP	Post EOP	Before FID	Notes
Post Manufacturing	No	Yes	Yes	Yes	Yes	On shared NVM HECI2 post EOS requires storage proxy.
Recovery	No	No	No	No	No	Not available.

WRITE DATA Request

Size Bytes	Field Name	Field Values	Description
32	HMAC-SHA256 Signature	HMAC-SHA256	The signature of entire data including counter but minus this field. Note: Place here to keep hashed fields contiguous for programmer ease.
1	Req_resp[0] Cmd_id[7]	0x04	Req_resp[0] 0'b – BIOS_TO_CSE Cmd_id[7] 0x2 – WRITE_DATA
4	Counter	0 – OxFFFFFFFF	The monotonic value from the host. CSE will confirm this is larger than the last MC value starting with the init value.
25	FilePath	ASCII	NULL terminated string in format <dirName/Filename>
4	Offset		Offset within file.
2	DataSize		Number of bytes to write.
1	Truncate		If set file will be truncated.
DataSize	Data		Data to write.

WRITE DATA Response

Size Bytes	Field Name	Field Values	Description
32	HMAC-SHA256 Signature	HMAC-SHA256	The signature of entire data including counter but minus this field. Note: Place here to keep hashed fields contiguous for programmer ease.



Size Bytes	Field Name	Field Values	Description
1	Req_resp[0] Cmd_id[7]	0x05	Req_resp[0] 1'b – CSE_TO_BIOS Cmd_id[7] 0x2 – WRITE_DATA
4	Counter	0 – 0xFFFFFFFF	CSE will return the request counter value.
1	Status	0x0 – 0xF	BIOS_HECI_STATUS_OK 0x0 BIOS_HECI_STATUS_INVALID_PARAM 0x1 BIOS_HECI_STATUS_FILE_NOT_FOUND 0x2 BIOS_HECI_STATUS_AFTER_EOS 0x3 BIOS_HECI_STATUS_DIR_LOCKED 0x4 BIOS_HECI_STATUS_NVM_INACCESSIBLE 0x5 BIOS_HECI_STATUS_TOO_MANY_DIR 0x6 BIOS_HECI_STATUS_COUNTER_INVALID 0x7 BIOS_HECI_STATUS_SIGN_INVALID 0x8 BIOS_HECI_STATUS_POST_EOP 0x9 BIOS_HECI_STATUS_POST_CORE BIOS_DONE 0xA BIOS_HECI_STATUS_NO_QUOTA 0xB BIOS_HECI_STATUS_ERROR 0xF

28.8.3 GET FILE SIZE

Availability

	Before DID	Post DID	Before EOP	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	Yes	
Post Manufacturing	No	Yes	Yes	Yes	Yes	
Recovery	No	No	No	No	No	Not available.

WRITE DATA Request

Size Bytes	Field Name	Field Values	Description
32	HMAC-SHA256 Signature	HMAC-SHA256	The signature of entire data including counter but minus this field. Note: Place here to keep hashed fields contiguous for programmer ease.
1	Req_resp[0] Cmd_id[7]	0x06	Req_resp[0] 0'b – BIOS_TO_CSE Cmd_id[7] 0x3 – GET_FILE_SIZE
4	Counter	0 – 0xFFFFFFFF	The monotonic value from the host. CSE will confirm this is larger than the last MC value starting with the init value.
25	FilePath	ASCII	NULL terminated string in format <dirName/Filename>

WRITE DATA Response

Size Bytes	Field Name	Field Values	Description
32	HMAC-SHA256 Signature	HMAC-SHA256	The signature of entire data including counter but minus this field. Note: Place here to keep hashed fields contiguous for programmer ease.
1	Req_resp[0] Cmd_id[7]	0x07	Req_resp[0] 1'b – CSE_TO_BIOS Cmd_id[7] 0x2 – WRITE_DATA
4	Counter	0 – 0xFFFFFFFF	CSE will return the request counter value.



Size Bytes	Field Name	Field Values	Description
1	Status	0x0 – 0xF	BIOS_HECI_STATUS_OK 0x0 BIOS_HECI_STATUS_INVALID_PARAM 0x1 BIOS_HECI_STATUS_FILE_NOT_FOUND 0x2 BIOS_HECI_STATUS_AFTER_EOS 0x3 BIOS_HECI_STATUS_DIR_LOCKED 0x4 BIOS_HECI_STATUS_NVM_INACCESSIBLE 0x5 BIOS_HECI_STATUS_TOO_MANY_DIR 0x6 BIOS_HECI_STATUS_COUNTER_INVALID 0x7 BIOS_HECI_STATUS_SIGN_INVALID 0x8 BIOS_HECI_STATUS_POST_EOP 0x9 BIOS_HECI_STATUS_POST_CORE_BIOS_DONE 0xA BIOS_HECI_STATUS_NO_QUOTA 0xB BIOS_HECI_STATUS_ERROR 0xF
4	FileSize		File size in bytes.

28.8.4 LOCK DIR

Availability

	Before DID	Post DID	Before EOP	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	Yes	
Post Manufacturing	No	Yes	Yes	Yes	Yes	
Recovery	No	No	No	No	No	Not available.

Lock DATA Request

Size Bytes	Field Name	Field Values	Description
32	HMAC-SHA256 Signature	HMAC-SHA256	The signature of entire data including counter but minus this field.

Size Bytes	Field Name	Field Values	Description
			Note: Place here to keep hashed fields contiguous for programmer ease.
1	Req_resp[0] Cmd_id[7]	0x08	Req_resp[0] 0'b – BIOS_TO_CSE Cmd_id[7] 0x4 – LOCK_DIR
4	Counter	0 – 0xFFFFFFFF	The monotonic value from the host. CSE will confirm this is larger than the last MC value starting with the init value.
25	FilePath		<Dir/Filename> Null terminated.

Lock DATA Response

Size Bytes	Field Name	Field Values	Description
32	HMAC-SHA256 Signature	HMAC-SHA256	The signature of entire data including counter but minus this field. Note: Place here to keep hashed fields contiguous for programmer ease.
1	Req_resp[0] Cmd_id[7]	0x9	Req_resp[0] 1'b – CSE_TO_BIOS Cmd_id[7] 0x4 – LOCK_DIR
4	Counter	0 – 0xFFFFFFFF	CSE will return the request counter value.



Size Bytes	Field Name	Field Values	Description
1	Status	0x0 – 0xF	BIOS_HECI_STATUS_OK 0x0 BIOS_HECI_STATUS_INVALID_PARAM 0x1 BIOS_HECI_STATUS_FILE_NOT_FOUND 0x2 BIOS_HECI_STATUS_AFTER_EOS 0x3 BIOS_HECI_STATUS_DIR_LOCKED 0x4 BIOS_HECI_STATUS_NVM_INACCESSIBLE 0x5 BIOS_HECI_STATUS_TOO_MANY_DIR 0x6 BIOS_HECI_STATUS_COUNTER_INVALID 0x7 BIOS_HECI_STATUS_SIGN_INVALID 0x8 BIOS_HECI_STATUS_POST_EOP 0x9 BIOS_HECI_STATUS_POST_CORE_BIOS_DONE 0xA BIOS_HECI_STATUS_NO_QUOTA 0xB BIOS_HECI_STATUS_ERROR 0xF

28.8.5 GET PROXY STATE

Availability

	Before DID	Post DID	Before EOP	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	Yes	Yes	
Post Manufacturing	No	Yes	Yes	Yes	Yes	
Recovery	No	No	No	No	No	Not available.

GET PROXY STATE Request

Size Bytes	Field Name	Field Values	Description
32	HMAC-SHA256 Signature	HMAC-SHA256	The signature of entire data including counter but minus this field. Note: Place here to keep hashed fields contiguous for programmer ease.

Size Bytes	Field Name	Field Values	Description
1	Req_resp[0] Cmd_id[7]	0x0A	Req_resp[0] 0'b – BIOS_TO_CSE Cmd_id[7] 0x5 – GET_PROXY_STATE
4	Counter	0 – 0xFFFFFFFF	The monotonic value from the host. CSE will confirm this is larger than the last MC value starting with the init value.

GET PROXY STATE Response

Size Bytes	Field Name	Field Values	Description
32	HMAC-SHA256 Signature	HMAC-SHA256	The signature of entire data including counter but minus this field. Note: Place here to keep hashed fields contiguous for programmer ease.
1	Req_resp[0] Cmd_id[7]	0xB	Req_resp[0] 0'b – BIOS_TO_CSE Cmd_id[7] 0x5 – GET_PROXY_STATE
4	Counter	0 – 0xFFFFFFFF	CSE will return the request counter value.



Size Bytes	Field Name	Field Values	Description
1	Status	0x0 – 0xF	BIOS_HECI_STATUS_OK 0x0 BIOS_HECI_STATUS_INVALID_PARAM 0x1 BIOS_HECI_STATUS_FILE_NOT_FOUND 0x2 BIOS_HECI_STATUS_AFTER_EOS 0x3 BIOS_HECI_STATUS_DIR_LOCKED 0x4 BIOS_HECI_STATUS_NVM_INACCESSIBLE 0x5 BIOS_HECI_STATUS_TOO_MANY_DIR 0x6 BIOS_HECI_STATUS_COUNTER_INVALID 0x7 BIOS_HECI_STATUS_SIGN_INVALID 0x8 BIOS_HECI_STATUS_POST_EOP 0x9 BIOS_HECI_STATUS_POST_CORE_BIOS_DONE 0xA BIOS_HECI_STATUS_NO_QUOTA 0xB BIOS_HECI_STATUS_ERROR 0xF

Details

Response will be sent when proxy is active.

28.9 BIOS to ISH Service Messaging

Fixed HECI client address ID is 0x03. Message format will be generic to allow future extendibility and contain command, message length, destination file name, destination file length and file content as data as well.

28.9.1 SET FILE

Availability

	Before DID	Post DID	Before EOP	Post EOP	Before FID	Notes
Manufacturing	No	Yes	Yes	No	Yes	
Post Manufacturing	No	Yes	Yes	No	Yes	



	Before DID	Post DID	Before EOP	Post EOP	Before FID	Notes
Recovery	No	No	No	No	No	Not available.

SET FILE Request

Size Bytes	Field Name	Field Values	Description
2	Command	0x01	For SetFile = 1
2	Message Length	N + 4	Maximum 8K + command message header
16	File Name		ASCII destination file name (up to not null terminated 12 bytes)
N	File Payload	See details	Content of the file

SET FILE Response

Size Bytes	Field Name	Field Values	Description
2	Command	0x01	For SetFile = 1
2	Message Length	0x04	4
4	Status	0 0xFFFFFFFF	0 – Success 0xFFFFFFFF – Failure

Details**28.9.1.1 PDT Unlock**

The feature uses ISH functionality.

BIOS communication m

The BIOS based unlock interface is used to implement PDT lock state protect from change by malicious SW. It is assumed that unlock command issued by BIOS provides certain protection level as it cannot be done by malicious SW.

BIOS does not need to create the PDT unlock message dynamically the message could be hardcoded into BIOS.

The only new functionality to be implemented by BIOS is trigger for sending the message to CSE before EOP and actual sending functionality (support for CSE file update protocol assumed to be already part of BIOS implementation).

The CSE shall implement changes in ISHA as described in previous section.

ISH shall implement support for bios2ish file retrieval and PDT unlock check based on presence of PDT_UNLOCK data type in the bios2ish file.



28.9.1.1.1 Flow

1. User uses BIOS menu or other BIOS specified method to request from BIOS to send "PDT Unlock" message to ISH as described in previous section.
2. BIOS sends the message to CSE and ISH retrieves it as described in previous section.
3. Host SW attempts to update PDT via ISH host interface
4. ISH PDT Update FW checks for presence of the "PDT Unlock" message in message from BIOS.
5. If "PDT Unlock" variable is present ISH PDT update FW accepts PDT update command similarly to situation where PDT was not locked.

The steps 3-5 of the flow might be performed multiple times until platform reset.

28.9.1.1.2 BIOS2ISH File with Single PDT_UNLOCK Message

Offset	Size (bytes)	Field Name	Value
0	2	Number of Elements	1
2	2	Data Type	1 – PDT_UNLOCK
4	2	Payload Size	0 – no data

28.9.1.2 PDT Update

The feature uses ISH↔BIOS communication mechanism to update PDT file in ISH.

BIOS should create the bios2ish file as a composition of the bios2ish header and the PDT file payload.

ISH will implement PDT update according to its inner logic (merge with old PDT file, replacement and so on.)

28.9.1.2.1 BIOS data push

1. Locate PDT entry in FOTA image.
2. Construct bios2ish file by appending PDT entry to header as defined later in this specification.
3. Communicate with CSE passing it the file.

28.9.1.2.2 BIOS2ISH file with single PDT_UPDATE request

Offset	Size (bytes)	Field Name	Value
0	2	Number of Elements	1
2	2	Data Type	2 – PDT_Update



Offset	Size (bytes)	Field Name	Value
4	2	Payload Size	Up to 4096
6	Up to 4096	Payload	Content of PDT file

28.10 Intel® Platform Trusted Technology

Intel® Platform Trust Technology is Intel implementation of TCG TPM 2.0 standard in firmware. Intel® PTT exists as a TXE FW application. The Host Controller Interface (HCI) provides a communication channel between Host Software and ME firmware.

Intel® PTT for Apollo Lake platform is also known as Intel® PTT Gen3.

Note: Intel® PTT is designed to meet Microsoft* Windows* 8/8.1/10 TPM2.0 requirements.

Note: Intel® PTT is supported on the version of Windows* 7 that supports TPM2.0.

Note: This document focuses on PTT specific features and BIOS requirements.

28.10.1 PTT Enabling

In order to utilize Intel® PTT feature, it is required to enable Intel® PTT in the FW image. The major option of PTT enable/disable is located in FPF (Field Programmable Fuse) or FPF Mirroring File if it's programmed. Setting in FPF can be checked by FPT (Flash Programming Tool) on target system. Setting in FPF Mirroring File can be found in IFWI by FIT (Flash Image Tool). **In addition to FPF there is also a soft strap and a BIOS enable/disable default setting.**

Note: If Intel® PTT is enabled, it will automatically disable the interface to dTPM, until Intel® PTT is disabled.

28.10.2 PTT vs. dTPM Detection

PTT is enabled, it will automatically disable the interface to dTPM, until PTT is disabled.

A platform can be designed to support both dTPM and PTT. However, only one of them can be active during each boot. IA FW should base on platform HW component and setting to enable or disable PTT for dTPM. The OEM must set dTPM location (LPC or SPI) during manufacturing.

28.10.3 TPM Commands

28.10.3.1 PKTPM Commands

PTT has a list of Pre-Kernel TPM commands. These commands are available shortly after _TPM_Init is sent by TXE.



Table 27-31. PKTPM Commands

Name	Supported Options
TPM2_EventSequenceComplete	All
TPM2_GetCapability*	Supported capabilities: TPM_CAP_ALGS TPM_CAP_TPM_PROPERTIES, with TPM_PT in PT_FIXED group (0x100-0x1FF) TPM_CAP_VENDOR_PROPERTY
TPM2_HashSequenceStart*	Only starting an Event sequence is supported
TPM2_HierarchyChangeAuth*	Only changing PlatformAuth supported
TPM2_HierarchyControl*	Only disabling Platform Hierarchy is supported
TPM2_PCR_Event	All
TPM2_PCR_Extend	All
TPM2_SelfTest	All
TPM2_SequenceUpdate	All
TPM2_Startup	All

Note: Rest of the TPM commands will be available 300ms after DID is sent. If a main TPM command is sent before it becomes available, it will not be rejected, but instead it will be postponed and no other commands will be processed until this command is processed.

Note: With PTT, the IA FW does not need to call TPM2_SelfTest.

TPM2_GetCapability*, TPM2_HashSequenceStart*, TPM2_HierarchyChangeAuth*, and TPM2_HierarchyControl* are only partially functional as the supported options indicated. However once reached MainTPM, they function as per TCG specification described. If any functionalities (parameters) not listed in supported options are sent during PKTPM, command will not be rejected, but instead will be postponed till MainTPM become available.

28.10.3.2 PTT Vendor Capabilities Values

Table 28-32. PTT Vendor Capabilities Values

Capability	Value
TPM_PT_MANUFACTURER	"INTC"
TPM_PT_VENDOR_STRING_1	"Inte"
TPM_PT_VENDOR_STRING_2	"I"
TPM_PT_VENDOR_STRING_3/4	""
TPM_PT_VENDOR_TPM_TYPE	0x00000002 (PTT)
TPM_PT_FIRMWARE_VERSION_1	Bits 16..31 – TXE FW Major version (0x0009)

Capability	Value
	Bits 0..15 – TXE FW Minor version (0x0005)
TPM_PT_FIRMWARE_VERSION_2	Bits 16..31 – TXE FW build number Bits 0..15 – TXE FW hot fix number

28.10.3.3 Failure Mode Reporting

While in failure mode, PKTPM may also support TPM2_GetTestResult instead of postponing it to MainTPM.

Note: In failure mode, only TPM2_GetCapability and TPM2_GetTestResult are available, all other commands will be responded with TPM_RC_FAILURE.

When any of the following errors occurs, PTT enters failure mode:

1. Fail to write to NV (when a failure was not expected)
2. Self-test failed / crypto failure
3. TXE reset without a host reset
4. Context load with bad sequence / context ID overflow
5. NV files: missing files / blob integrity failure / AR failure
6. Total Reset counter overflow
7. Memory allocation failure
8. SRAM Parity error
9. System errors (ThreadX objects creation failure, and so on.)
10. Fuses error (EK not fused correctly or FPF read/write operation failed)

When PTT is in failure mode, the command TPM2_GetTestResult will return the failure mode reason and other information in the vendor-specific area. There may be more than one reason listed.

The vendor-specific area is parsed as follows:

NumArgs for failure description #1
Arg 1 - Failure description #1
Arg 2
...
NumArgs for failure description #2
Arg 1 - Failure description #1
Arg 2
...



Below Table lists the possible values for the failure descriptions. Note that the argument parsing is different for each failure.

“Possibly recoverable errors” are errors that may be solved by a global reset and do not require a platform recall.

Table 28-33. Reason ID in TPM2_GetTestResult

	Description	Possibly recoverable error?
0x00000000	no failure	n/a
0x0000001B	PTT was reset without a host reset	Yes – global reset required
0x0000001C	Context load with bad sequence	Yes – global reset required
0x0000001D	Context ID overflow	Yes – global reset required
0x0000001E	Total reset counter overflow	No – reflash may solve this.
Other errors between 0x00000001-0x000000FF	General failures	Unknown – global reset may resolve this.
0x00000100-0x000001FF	Hash-related errors	Unknown – global reset may resolve this.
0x00000200-0x000002FF	TXE internal errors	Unknown – global reset may resolve this.
0x00000300-0x000003FF 0x00000600-0x000006FF	NV related errors	Global reset <i>may</i> resolve this (TPM may be cleared). Otherwise a reflash will be required.
0x00000400-0x000004FF	RSA related errors	Unknown – global reset may resolve this.
0x00000500-0x000005FF	Fuses (EK fuses or FPF) errors	No
0xFFFFFFFF	unknown reason	This error shows up when a failure mode occurred and then an S3 resume occurred.

28.10.3.4 Main TPM commands

300 ms after DID is sent, rest of the Main TPM commands will become available. Below is a list of them. All PKTPM commands will continue to be available and included in the table below.

Any commands not listed below are not supported by PTT.

Table 28-34. Main TPM Commands

Commands	
TPM2_ActivateCredential	TPM2_ObjectChangeAuth
TPM2_Certify	TPM2_PCR_Event
TPM2_CertifyCreation	TPM2_PCR_Extend
TPM2_ChangeEPS	TPM2_PCR_Read
TPM2_Clear	TPM2_PCR_Reset
TPM2_ClearControl	TPM2_PolicyAuthorize
TPM2_ClockSet	TPM2_PolicyAuthValue
TPM2_ContextLoad	TPM2_PolicyCommandCode
TPM2_ContextSave	TPM2_PolicyCounterTimer
TPM2_Create	TPM2_PolicyCpHash
TPM2_CreatePrimary	TPM2_PolicyDuplicationSelect
TPM2_DictionaryAttackLockReset	TPM2_PolicyGetDigest
TPM2_DictionaryAttackParameters	TPM2_PolicyNameHash
TPM2_Duplicate	TPM2_PolicyNV
TPM2_EventSequenceComplete	TPM2_PolicyOR
TPM2_EvictControl	TPM2_PolicyPassword
TPM2_FlushContext	TPM2_PolicyPCR
TPM2_GetCapability*	TPM2_PolicyRestart
TPM2_GetRandom	TPM2_PolicySecret
TPM2_GetTestResult	TPM2_PolicySigned
TPM2_Hash	TPM2_PolicyTicket
TPM2_HashSequenceStart*	TPM2_Quote
TPM2_HierarchyChangeAuth*	TPM2_ReadClock
TPM2_HierarchyControl*	TPM2_ReadPublic
TPM2_Import	TPM2_RSA_Decrypt
TPM2_Load	TPM2_SelfTest
TPM2_LoadExternal	TPM2_SequenceComplete
TPM2_MakeCredential	TPM2_SequenceUpdate
TPM2_NV_ChangeAuth	TPM2_Shutdown
TPM2_NV_DefineSpace	TPM2_Sign
TPM2_NV_Increment	TPM2_StartAuthSession
TPM2_NV_Read	TPM2_Startup
TPM2_NV_ReadPublic	TPM2_StirRandom
TPM2_NV_UndefineSpace	TPM2_Unseal



Commands	
TPM2_NV_Write	TPM2_VerifySignature

TPM2_GetCapability*, TPM2_HashSequenceStart*, TPM2_HierarchyChangeAuth*, and TPM2_HierarchyControl* are fully functional per TCG specification described at MainTPM stage.

28.10.4 PTT Initialization

When system powers on, TXE will issue the _TPM_Init signal to PTT. This enables the PKTPM commands. For the purpose of the components targeted by this document, execution should proceed to TPM Startup. Before sending the TPM2_Startup command, IA FW needs to determine if PTT or dTPM should be used. Refer Section [28.7.2](#). Once IA FW determined that PTT should be active on the platform, IA FW should follow TCG TPM 2.0 Spec to send the TPM2_Startup command.

If this is a system restart/s4/power on, startupType value CLEAR should be issued. If a system is coming out of S3, then IA FW should call TPM2_Startup with startupType value set to STATE.

Note: IA FW should publish the TPM2 ACPI Table if PTT is enabled. If PTT is enabled, but not ready, meaning there is a PTT error. Platform should still publish the TPM2 ACPT Table which will result in a yellowbang in OS. Purpose is that end-user/OS will get notified there is an error and take action accordingly.

If PTT interface is made inaccessible or disabled, then dTPM interface will be visible when dTPM is physically present on the platform. Platform IA FW should be aware of this and act accordingly.

28.10.4.1 Error during S-CRTM PTT Initialization for S4/S5

If the S-CRTM is unable to successfully initialize the PTT using the TPM2_Startup (CLEAR) command, it is possible that later code could successfully issue the command and record false integrity measurements. To prevent this, the S-CRTM takes steps to prevent use of PTT or platform if it is unable to successfully initialize PTT. A special case is when the attempt to initialize PTT results in PTT being in failure mode; later components will be unable to record false integrity measurements because for most commands PTT will continue to return TPM_RC_FAILURE.

Below shows the remediation steps the S-CRTM takes when PTT initialization fails.

Note: This applies to S4/S5 or normal boot.

If PTT is accessible and one of the following situations occurs:

1. If the S-CRTM is unable to successfully issue the TPM2_Startup(CLEAR) command, OR
2. The S-CRTM issues the TPM2_Startup(CLEAR) command and the return result does not equal TPM_SUCCESS, TPM_RC_INITIALIZE, or TPM_RC_FAILURE,
 - a. Then the S-CRTM must meet the TCG PC Client Specific Implementation Specification for Conventional IA FW requirements which are: Make the PTT interface inaccessible. Refer Section [28.7.8](#).



Note: Disabling PTT requires a platform reset.

1. Reboot the Host Platform; Or
2. Disable the Host Platform.
 - a. If the S-CRTM issues the TPM2_Startup(CLEAR) command and return result equal TPM_SUCCESS, TPM_RC_INITIALIZE, or TPM_RC_FAILURE, IA FW shall continue to boot. However when TPM_RC_FAILURE is returned, it is recommended not to send TPM commands anymore.

28.10.4.2 Error during S-CRTM PTT Initialization for S3

If the S-CRTM is unable to successfully initialize the PTT using the TPM2_Startup (STATE) command, it is possible that later code could successfully issue the command and record false integrity measurements. To prevent this, the S-CRTM takes steps to prevent use of PTT or platform if it is unable to successfully initialize PTT. A special case is when the attempt to initialize PTT results in PTT being in failure mode; later components will be unable to record false integrity measurements because for most commands PTT will continue to return TPM_RC_FAILURE.

Below shows the remediation steps the S-CRTM takes when PTT initialization fails.

Note: This applies to S3 resume.

If PTT is accessible and one of the following situations occurs:

1. If the S-CRTM is unable to successfully issue the TPM2_Startup(STATE) command, OR
2. The S-CRTM issues the TPM2_Startup command and the return result does not equal TPM_SUCCESS, TPM_RC_INITIALIZE, or TPM_RC_FAILURE.

Then the S-CRTM must reboot the Host Platform to protect TPM assets.

28.10.5 Extending PCRs

IA FW may use either TPM2_PCR_Extend or TPM2_PCR_Event to extend values into PCRs. TPM2_PCR_Extend requires the caller to first hash the data which will be extended in the PCR, while TPM2_PCR_Event takes the data and hash by PTT before extending to PCRs. TPM2_PCR_Extend allows for increased performance while TPM2_PCR_Event allows for increased algorithm agility.

If the caller chooses to use TPM2_PCR_Extend, the caller needs to be sure it uses hash algorithms that are supported by the TPM. The caller can know the correct hash algorithm either by having prior knowledge of the PPT (e.g., current PTT only supports SHA-1 and the IA FW may know that) or by using TPM2_GetCapability.

Note: It is recommended to use TPM2_PCR_Extend for the performance benefit.

Note: IA FW must follow TCG specifications, TPM application requirements and OS vendors requirements for what are expected for each PCRs. And EventLog creations.



28.10.5.1 Error Recording SRTM Measurement

If the measurement of the SCRTM, POST IA FW or Embedded Option ROMs cannot be made, then the S-CRTM must meet the *TCG PC Client Specific Implementation Specification for Conventional IA FW* requirements. The S-RTM MUST be capped by extending value of 0x01 to each PCR[0-7]. This S-RTM should log the error event to the event log. IA FW should continue on in this error case.

If each PCR[0-7] cannot be capped, the Host platform should take any necessary action to notify the Host Platform's administrator, user, and operator along with transitioning into a "fail-safe" mode by performing one of the these actions:

Make the PTT interface inaccessible. Refer [28.7.8](#).

Note: Disabling PTT requires a platform reset.

1. Reboot the Host Platform; Or
2. Disable the Host Platform.

28.10.6 Enabling/Disable Hierarchies and Establish Random Value for PlatformAuth

After a successful TPM2_Startup, IA FW may choose to enable/disable each of the Hierarchies and must establish a random value for PlatformAuth.

Platform, Storage, and Endorsement Hierarchies are all reset to be enabled on every normal boot. During normal TPM operation, those hierarchies should remain enabled.

IA FW may disable/enable Platform, Storage and Endorsement Hierarchies by using TPM2_HierarchyControl which requires PlatformAuth.

Note: PlatformAuth value is reset to 0 on every boot until PlatformAuth is set. And PlatformAuth value will be maintained during S3.

IA FW must create a cryptographically strong random value for the PlatformAuth on each boot or disable Platform Hierarchy if a new PlatformAuth value is not created.

The maximum size for this value is the length of the hash produced by the algorithm used for context integrity, which may be determined by using TPM2_GetCapability. This value is then put into the TPM using TPM2_HierarchyChangeAuth.

Note: IA FW should iteratively use the processor RDRAND instruction to obtain a random value with the maximum number of bytes supported for the PlatformAuth.

After setting the PlatformAuth, the IA FW must either securely store the PlatformAuth in memory or scrub the value from memory (overwrite with 0's). The PlatformAuth value must be available in order to enable/disable hierarchies.

If IA FW is going to disable the Platform Hierarchy, setting a new PlatformAuth is not required. If Platform Hierarchy is disabled, any tpm commands that require PlatformAuth cannot be used, including TPM2_HierarchyControl which is needed to enable/disable Storage and Endorsement Hierarchies.

Note: Choosing Platform Hierarchy disable requires the disabling action to be done on every Sx exit as Platform Hierarchy will be re-enabled on every Sx entry.

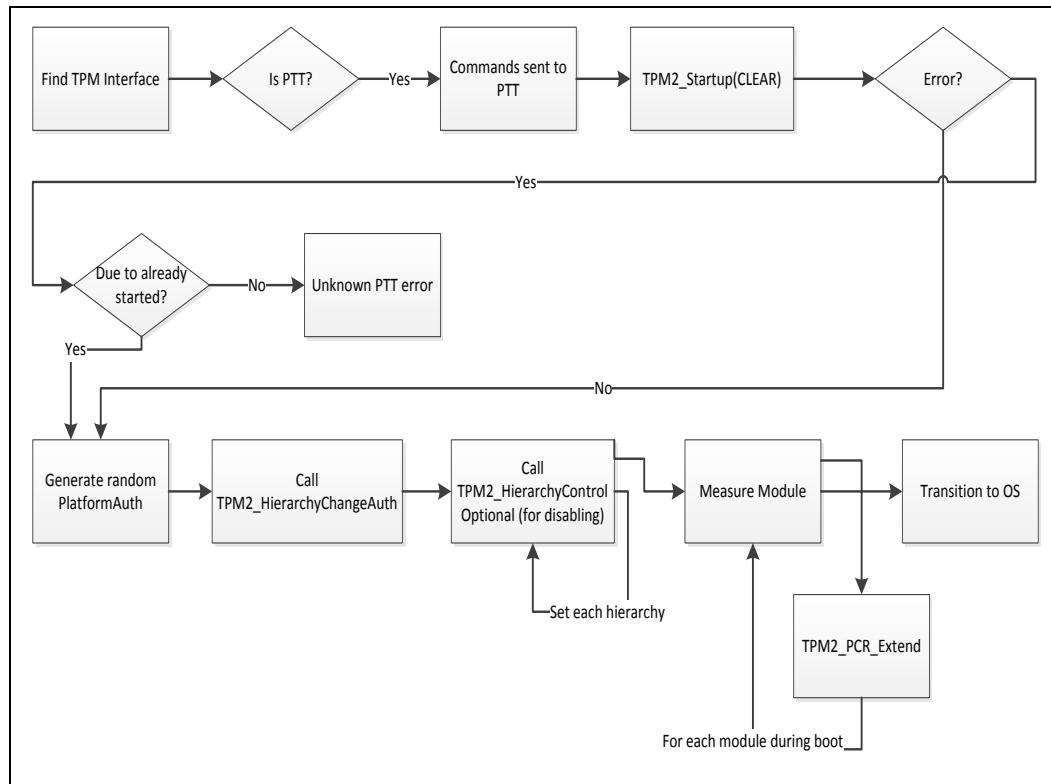
IA FW may choose to disable Platform Hierarchy and set a new PlatformAuth any time prior to exitPmAuth (before any third party software/oprom start executing).

28.10.6.1 Recommendation for Platform Hierarchy/PlatformAuth Usage

If IA FW does not need to use Platform Hierarchy during Runtime or S3 resume, then simply set a new PlatformAuth would be sufficient. There is no need to securely store the PlatformAuth value as IA FW is done using Platform Hierarchy during POST and do not need to remember the PlatformAuth during Runtime or S3 resume.

There is a need to use Platform Hierarchy during Runtime or S3 resume, then IA FW must securely store PlatformAuth at a location such as SMRAM so it can be securely retrieved.

Figure 28-3. Flow Diagram



28.10.7 Enabling/Disabling PTT during POST

To enable/disable PTT during POST, IA FW must first check PTT capability via GET FW CAPABILITIES message to ensure platform is PTT capable, then IA FW should check PTT Enable Bit to see the current PTT status, then it can send the FW FEATURE STATE OVERRIDE message to TXE FW to enable/disable PTT accordingly.



Bit29 is PTT for both the Enable and Disable Bit Mask.

Note: Global Reset is required after the FW FEATURE STATE OVERRIDE message is accepted by the Intel® TXE firmware.

If a setup option is made available, then when end users select the option to disable PTT in IA FW setup, they should be asked to confirm the action with the warning of the impact. They may also be given the options on whether they want to perform TPM2_Clear with proper description on the impact as well.

A sample execution flow would be:

1. Call TPM2_ClearControl - Optional
2. Call TPM2_Clear - Optional
3. Send FW FEATURE STATE OVERRIDE message
4. Issue platform reset

28.10.8 Switching between PTT and dTPM1.2 during POST

To switch from PTT to dTPM1.2, IA FW only need to disable PTT, then dTPM1.2 Interface will be visible on the next boot. However if a setup option is made available, before end-user select the option to switch from PTT to dTPM1.2, a warning of the action impact should be displayed. They may also be given the options on whether they want to perform TPM2_Clear along with proper description on the impact as well.

A sample execution flow would be:

1. Call TPM2_ClearControl - Optional
2. Call TPM2_Clear - Optional
3. Send FW FEATURE STATE OVERRIDE message
4. Issue platform reset

Switching from dTPM1.2 to PTT should follow the same recommendation as above. However dTPM1.2 commands will be different.

Before sending commands, IA FW should check PTT capability, then PTT enable/disable state then act accordingly.

28.10.9 Re-Provisioning

To re-provision PTT, Endorsement, and Storage Hierarchies should be cleared. PTT does not support clearing Platform Hierarchy. IA FW should implement two separate options to perform TPM2_Clear and ChangeEPS. IA FW should clearly explain each option and ask for confirmation from end-user before proceed.

Note: PlatformAuth is required for each action.

28.11 SPI Less EFI Runtime Storage

UEFI spec recommends EFI Variables services which demands NVM storage space for some of the platform use cases, spec provides APIs to access this NVM space during



boot time and runtime context. But NVM space usage during runtime entirely depends on the OS use cases and some of the OEM Platform use cases. As of the current data, Windows* has quite a few use cases that demands runtime access to this EFI NVM data area and Android* has very limited use cases that demands runtime access NVM data.

BRX support single headed eMMC/UFS storage i.e. means at any given time one entity i.e. either TXE or Host can own the storage head. This is not an issue till the OS calls Exitbootservice API to end the EFI boot time API usage, in this space TXE and BIOS can exchange the Storage head ownership based on the need.

28.11.1 NVM Requirements

1. BIOS must have dedicated NVM storage space.
 - Android* NVM storage space which needs runtime access: 128 Bytes
 - Windows* storage requirement: 64KB
 - Platform storage requirement that is common for both OS – 128KB
 - MRC/Platform configuration/manufacturing data and so on.
 - OEM are – up to OEM could be another 256KB.
2. NVM must be accessible during all phases of platform boot.
 - IBB
 - OBB to exit boot service.
 - OS driver owning the storage head.
 - Still in OS context where OS storage driver is not available.
 - Or during system user initiated reboot/shutdown phase.
3. NVM storage must not lock up a storage device to the platform till the platform leaves the manufacturing floor.
4. NVM solution must not take away more than 1MB of storage space from platform storage media.
5. Security.
 - Storage of secure data must be isolated from the running operating system such that they cannot be modified without detection.
 - UEFI API's must persist the values on NVM before returning SUCCESS.
 - NVM access must be secured and attack NVM must at least be able to boot the system to UEFI phase where one can perform manual recovery to re-flash the OS images.

28.11.2 NVM Access Flow

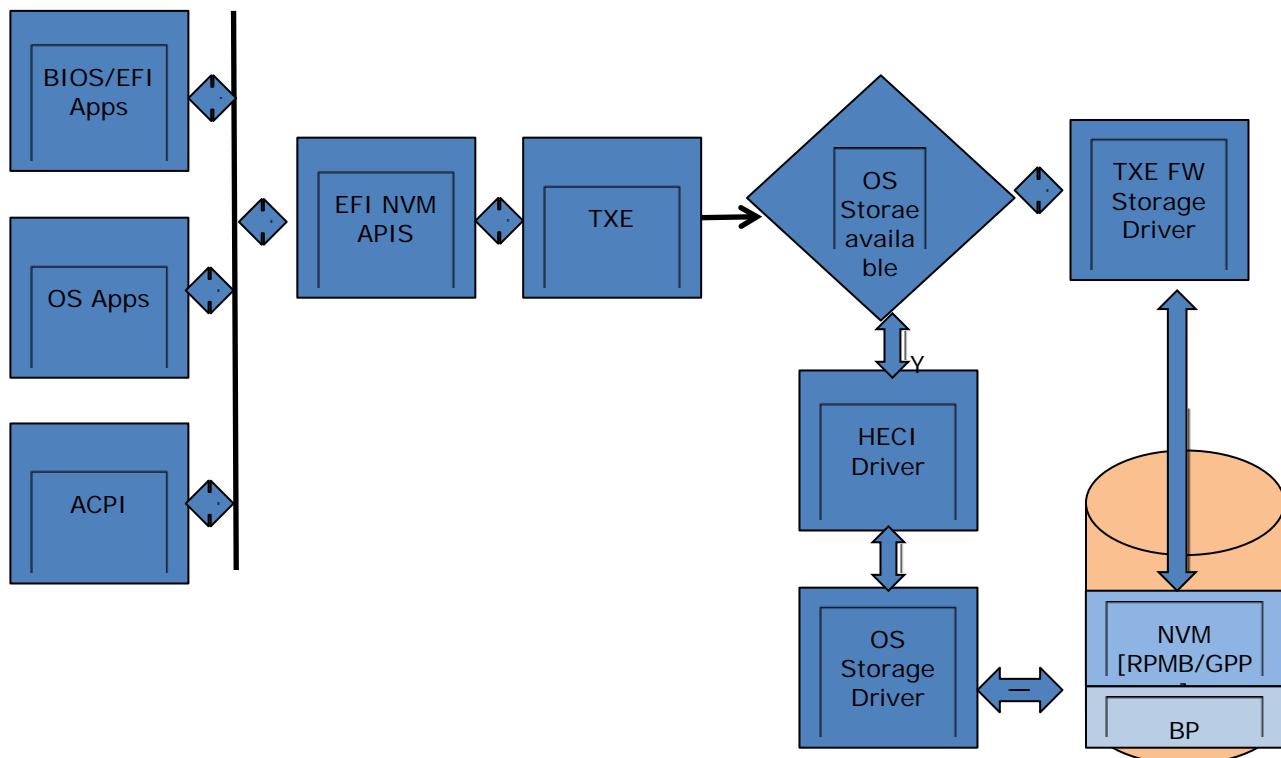
Using the RPMB [GPP as pre manufacturing] as the single NVM storage for BRX. Below summarizes the overall NVM Flow.

1. TXE and BIOS are using one common NVM storage partition of eMMC or UFS, secure space is coming from RPMB for production with GPP usage for pre-manufacturing.
2. RPMB as NVM solution is ideal from the point of view of security and ideal form the point of view of limited user space usage point of view.

3. Slight complications in the overall manufacturing process as early usage of RPMB binds a specific Emmc/UFS part to the platform and this early locking of part to platform is not liked by OEM as they cannot move around the parts easily or it does not allow them recover from RPMB key provision err.
 - a. Early usage of RPMB under manufacturing process could be avoided by the usage of EMMC General Partition as the NVM space under manufacturing.
 - b. An Android* OEM who wishes to remove RPMB dependencies on production, could continue with GPP as the NVM space. On Windows* RPMB usage is mandatory as FTPM needs a space which is physical attack resistance.

Overall BIOS NVM flows Solution can be classified into 2 types.

1. TXE directly performing NVM storage access. This would cover the below 3 phases of the BIOS NVM flows.
 - a. BIOS IBB phase using NVM –pre DRAM init less scenario.
 - b. BIOS later phase, till Exitbootservice performing NVM access .BIOS performing NVM transactions when system reboot/shutdown phase is in progress.
2. OS driver owning the storage head, TXE relying on OS storage driver to perform the NVM transactions.
 - a. Most of the runtime access happen here with the exception of OS Storage driver crash **case scenario**.



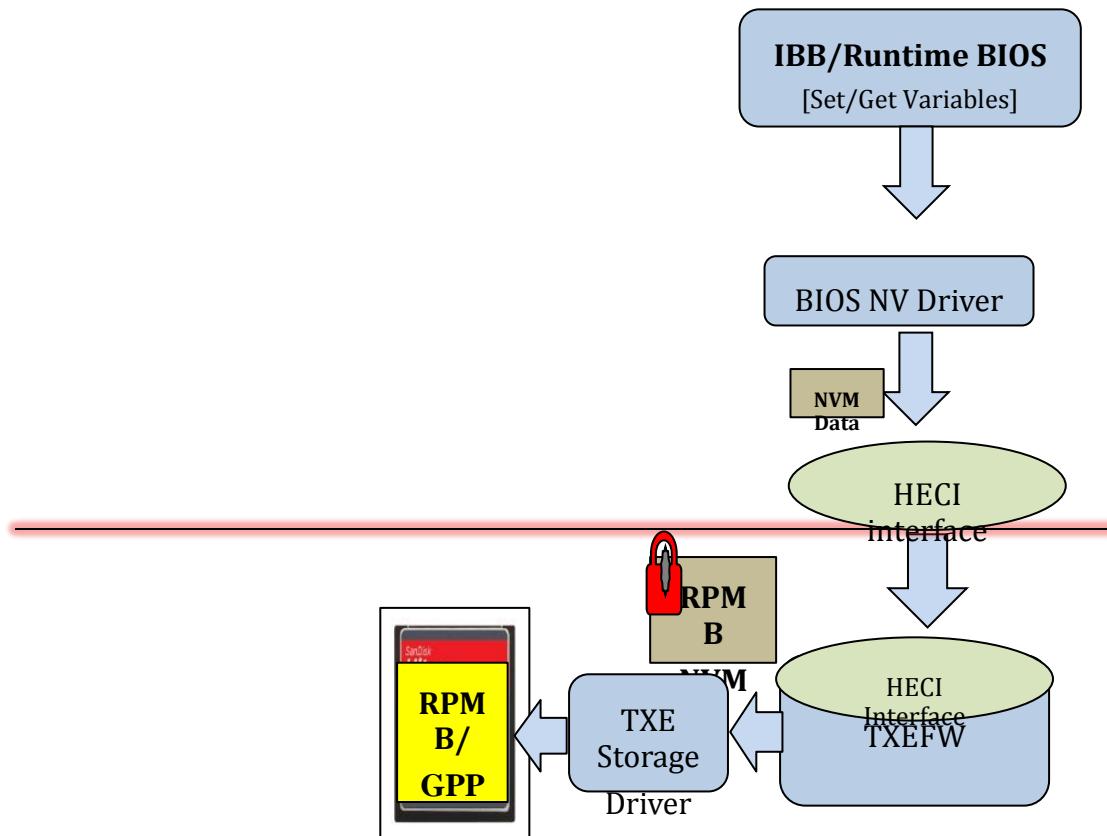
28.11.2.1 TXE Direct NVM Access

This is the scenario that could come into picture during one of the below listed scenario.

- BIOS IBB phase using NVM –pre DRAM init less scenario.
- BIOS later phase, till Exitbootservice performing NVM access.
- BIOS performing NVM transactions when system reboot/shutdown phase is in progress.

Scenario-a happen without SMM NVRAM or BIOS NVRAM DXE driver being loaded and could happen either through DRAM less environment or early phases of the DRAM based environment where BIOS DXE NVM storage is not fully available. BIOS could still make use of the storage access HECI Message described in the later in the document to perform NVM tractions via TXE.

During this stage TXE does not have to enforce the SMM SAI protection check on the NVM R/W HECI message that BIOS sends. Traditional BIOS only performed NVM Read access in this phase.





Flow Summary

- TXE control the storage and till memory init, BIOS to use CASE SRAM space to handle the perform storage R/W –with a max limitation of 4KB
- BIOS to manage the NVM space – BIOS to manage the mapping of UEFI variable to RPMB NVM address space.
- BIOS to use HECI message to TXE to perform NVM R/W access.
- BIOS to provide TXE SRAM [from IBB] or system memory to perform read/write to NVM space.
- Before MRC – BIOS to perform NVM read/write to CPU Cache and make use of it, SRAM space allotted could be a max of 4KB.
- Post memory init no restriction on R/W size but at any given time its advised to restrict the NVM access at 64KB boundary.
- Used by IBB initially and IBB to use Storage R/W HECI MSG.
- When UEFI Runtime API gets called, in the absence of OS storage driver availability – TXE to perform direct NVM transactions on behalf of BIOS HECI message.

28.11.3 NVM under TXE Control – SMM Managing NVM Transactions

- This is the scenario that could come into picture during one of the below listed scenario.
- BIOS later phase, till Exitbootservice performing NVM access.
- BIOS performing NVM transactions when system reboot/shutdown phase is in progress.
- Key difference is that BIOS would load/launch full DXE NVM services by now and SMM driver would manage the crypto/variable parsing /Variable NVM space to RPMB address mapping. SMM also would issue a HECI command, ensure TXE acknowledges HECI command from SMM SAI, and come out of SMM to wait for the NVM R/W response from TXE.
- Upon receiving the EOP HECI message, TXE must enforce SMM SAI check on HECI to ensure that RPMB R/W message indeed has come from the BIOS SMM handler.
- Overall sequence outlined here and the corresponding sequence diagram is outlined in the subsequent section.
- BIOS apps/OS apps/BIOS later phase code /BIOS reboot/shutdown API to invoke UEFI set/get services API.
- BIOS to pass the NVM parameter to SMM handler and trigger Software SMM.
- SMM handler to perform the following tasks.
- NVM parameter check.
- Variable crypto processing to ensure secure variables are authenticated, BIOS could use the same HECI head to offload the crypto services like RSA to TXE to reduce the SMM handler code size.
- SMM handler to find out the Variable mapping to the RPMB address space.
- SMM to issue the HECI message, wait for the response.



- If HECI result is FAIL –set NVMOPCompelete = TRUE exit NVModelErrorcode= TXE returned error and exit the SMM.
- If HECI result is SCUCESS – NVMOPCompelete= FALSE and exit SMM
- come to the Get/Set variable context:
- If NVMOPCompelete = TRUE then return the SET/GET API with the NVModelErrorcode= TXE returned error.
- If NVMOPCompelete= FALSE then wait till it becomes true as TXE is still processing the NVM commands.
- TXE to receive the NVM R/W HECI message.
- TXE to enforce SAI check on SMM if EOP is already received from BIOS.
- SMM SAI check failed –BIOS to return fail to R/W Storage HECI MSG.
- If SMM SAI check is passed, TXE to return SUCCESS to R/W Storage HECI MSG
- TXE to check if OS storage driver is available [by default not available, HECI OS driver/ACPI to coordinate to update TXE on OS storage driver availability.
- If OS driver is not available:
- TXE to switch the storage head to TXE address map.
- TXE to perform RPMB partition switch.
- If NVM operation is WRITE, TXE to perform data blob signing.
- TXE to perform the requested NVM operation.
- TXE to trigger an SMM.
- BIOS SMM handler gets called.
- SMM to issue HECI message to read the NVM data from UMA area via HECI MSG.
- TXE to return the status.
- If status is success and NVM was a read operation:
- TXE to return the requested data if NVM Read operation is performed or else.
- SMM to set the NVMOPCompelete memory address = TRUE.
- BIOS Get/Set API which is spinning on NVMOPCompelete to get the result and return; before returning BIOS set the storage head to Host address map, perform partition context switch.

28.11.3.1 NVM Access under OS Control

This scenario covers the UEFI Get/Set variable access during the OS storage driver present case. When the OS storage driver gets the control, it takes over the exclusive ownership of the storage head. There are 2 ways BIOS/TXE interaction could happen under this scenario.

- BIOS Runtime NVM driver directly interacting with TXE to perform the secure NVM via TXE, TXE to reuse its secure NVM flow via OS secure channel driver. Here the assumption is that exclusive HECI head is accessible for BIOS/TXE interactions.



BIOS runtime NVM driver, piggybacking on TXE HECI driver to perform NVM transactions. HECI driver to interfacing with OS storage to perform BIOS NVM transactions.

28.11.3.2 Runtime NVM with Exclusive HEC Head for BIOS SMM Usage

Here we are assuming exclusive HECI Header availability for BIOS [SMM]-TXE interactions. From security perspective/flow simplicity perspective this option is better. BRX has 3 HECI heads, Windows* with VMM would use 2 and one HECI head is still free for BIOS NVM usage. With Android* Silent Pass is taking away 1 HECI head and leaving no separate HECI HEAD for BIOS NVM usage. Subsequent BRX derivative we could ask for an extra HECI head.

Even for NVM happening OS control, BIOS NVM flows listed would still be reused, only difference is would be at TXE level. TXE FW knowing the HECI driver presence would push the messages upstream to perform NVM R/W. Once the result is available, TXE would trigger an upstream SMI and let the BIOS SMM handler to grab the result, set the flag on which UEFI Runtime is spinning for TXE Storage operation completing status.

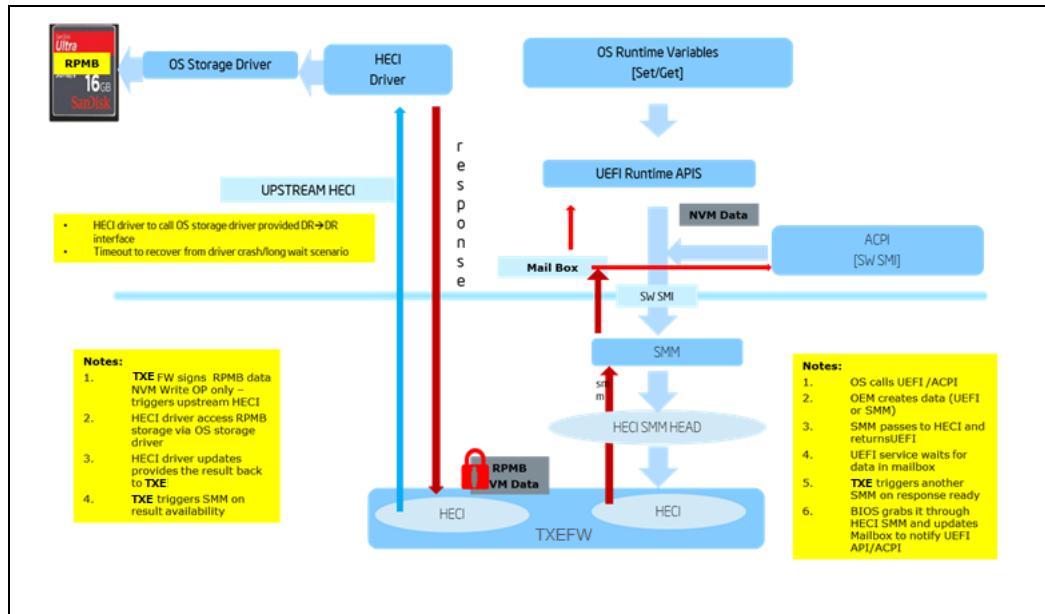
TXE is to reuse the flows of its Runtime NVM access via the channel secure driver which is HECI driver talking to OS storage driver to perform the NVM R/W access.

TXE is to sign the NVM Write data with the RPMB Key and poke an upstream HECI message. For NVM read operation, there is not any signing requirement and TXE to reuse its NVM flow to perform the read via the HECI driver and get the response and trigger an SMM. SMM handler getting the result via HECI messaging and setting the Flag on which EFI runtime API would spinning to wait the for TXE response.

1. OS to call EFI runtime API, which would trigger SW SMM related to secure NVM
2. BIOS SMM NVM driver explained in the previous section would be reused– BIOS SMM handler to issue HECI MSG, ensure TXE gets in SMM SAI and return to UEFI Runtime API context and poll on **NVMOPComplete**.
3. TXE receiving the BIOS NVM HECI message, before processing must have to ensure its coming from SMM handler by performing the SMM SAI check. If SMM SAI check fails, TXE to drop the BIOS NVM HECI MSG and return error as part of BIOS NVM HECI MSG.
4. On SMM SAI check being passed and to check if OS HECI driver is presents and not voluntarily unloaded by OS, if so TXE to return error to the BIOS NVM HECI MSG. if driver is available, TXE return SUCCESS as a response to BIOS NVM HECI MSG and start processing the BIOS NVM HECI MSG.
5. If BIOS is requesting NVM write and NVM storage is RPMB, TXE to sign the data BIOS NVM Data blob signing.
6. TXE to reuse its own runtime to flow get the BIOS NVM R/W operation performed by triggering an upstream HECI MSG.
7. TXE HECI driver to decipher the NVM HECI message and to see what kind of NVM operations has been requested. Based on the type of operation, HECI driver to issue R/W operations to the OS storage driver.
8. HECI driver to provide the operation status and operation result back to TXE via downstream HECI MSG.

9. Upon receiving the HECI MSG response from OS HECI driver, TXE FW to reuse the previously described flow of triggering SMM to push the result back to UEFI Get/SetVariable context.
10. SMM handler to grab the result via HECI and return back to UEFI Runtime context by setting the NVMOCPComplete=TRUE and also SMM to fill the read data on UEFI GetVariable provided buffer.
11. UEFI Set/Get API to get exit the call.

Figure 28-4. Flow Listed below Assumed BIOS Controllable HECI HEAD



28.11.3.3 HW

- HECI SMM head: reuse th3 Silent Lake SMM Head, not an issue for Windows* as 3rd HEAD is available.
- SEC ability to trigger an SMM to provide response back.
- UEFI Runtime API to trigger an SCI/GPIO Pin to invoke an ASL handler.

28.11.4 TXE

- Provide NVM address space on RPMB -512KB max.
- Enable HECI commands for BIOS NVM data access on RPMB/GPP.
- From EOP stage ensure CORE is in SMM before processing NVM access HECI command.
- Perform RPMB/GPP R/W by reclaiming the storage ownership every time BIOS issues NVM R/W HECI MSG and OS Storage driver is not loaded.
- Perform NVM Storage access via OS storage driver by coordinating with Intel secure channel driver i.e. HECI driver.



28.11.5 BIOS

- UEFI Runtime API with MailBox to poll for TXE provided RPMB access result via an SMM
- SMM handlers –must ensure all CORES into SMM.
- SMM based HECL interface.
- SMM manage the NVM space on RPMB /GPP.
- SMM to Push NVM access to TXE, Come out of SMM to ensure TXE could make use of OS infrastructure.
- SMM to pick the result through HECL triggered SMM, update the Mailbox to indicate UEFI NVM/ACPI on result availability.
- Manage SMM HECL head D3/D0 transition.

28.11.6 HECL Driver

- Inform the TXE about driver availability/unavailability.
- Intercept TXE triggered upstream RPMB messages, call OS storage driver enabled driver → driver RPMB access APIS.
- Provide the result back to TXE.

28.11.7 OS Storage Driver

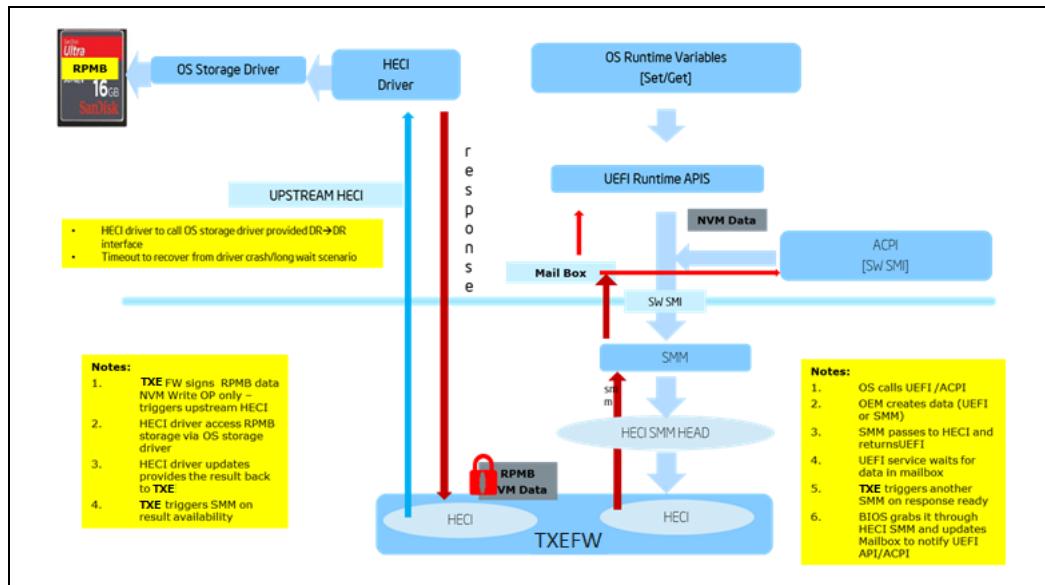
- Provide RPMB access storage APIS –Driver to driver interfacing.

28.11.8 ACPI

- RTD3 support on TXE HECL devices.
- On triggering NVM access via SW SMM- has to poll on MailBox for result availability.
- Get back the result through memory.

28.11.9 BIOS Performing NVM Transactions via TXE HECL Driver

Key difference from previous case is that there is not a HECL head available for BIOS SMM to interact with TXE in an OS isolated environment. Thus the only way for BIOS runtime API is to go upstream to the OS HECL driver to perform the NVM. This scheme is key if in case BIOS does not get separate HECL head and need to enable common runtime NVM scheme between AOS and WOS.



28.12 TXE Temp Disable

TXE can be temporarily disabled for system debug and servicing. In these cases only the PKTPM commands will be available during boot. OS will not have access to PTT at all. And IA FW will have access to limited API during boot time.

Note: This feature is used by the OEM/ IT for the debug purposes and should not be accessible by the platform end user.

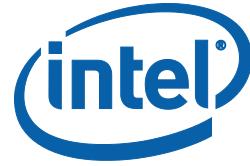
BIOS can utilize the “SET ME DISABLE” and “SET ME ENABLE” messages to switch the TXE firmware back and forth between temporary disable mode and normal mode.

28.13 GPP1 Lock

GPP1 partition in BIOS is lock down (Power-on write protect) the partition in the OS loader just before handing off to the OS. With such a change, any rooted device or any OS malware even with root privileges will not be able to alter GPP contents. This has also the advantage to allow the bootloader to access the partition for provisioning purpose as provisioning is supported.

The GPP1 power-on write protect is done only if the device is in EndUser configuration. In manufacturing, the partition will not be locked as we need R/W access from OS.

§



Appendix A MONITOR and MWAIT

A.1 MONITOR and MWAIT

MONITOR and MWAIT are provided to improve synchronization between multiple agents. MONITOR defines an address range that is used to monitor write-back stores at addresses within that range. MWAIT is used as a hint to indicate that the software thread is waiting for a write-back store to the address range defined by the MONITOR instruction.

The support for MONITOR and MWAIT instructions must be detected by the MONITOR/MWAIT bit in the CPUID feature flags. While the MONITOR and MWAIT instructions are supported in the processor, they may be explicitly disabled either by the OS or the IA FW. Disabling the instruction clears the CPUID feature flag and causes execution of the MONITOR or MWAIT instruction to generate an illegal Opcode exception. Software need not check for support of SSE in order to use the MONITOR and MWAIT instructions.

The processor uses extensions and hints for MWAIT to effectively enable the use of MONITOR and MWAIT instructions as native ACPI defined C-state instructions.

A.1.1 Monitor-MWAIT Address Range Determination

Software should know the exact length of the region that will be monitored for writes by the Monitor-MWAIT instructions. Allocating and using a region smaller in length than the triggering area for the processor could lead to false wake-ups resulting from writes to data variables that are incorrectly located within the triggering area. Conversely, allocating and using a region greater in length than the triggering area for the processor could lead to the processor not waking up after a write intended for the triggering area. The CPUID instruction provides a mechanism for determining the exact length of the triggering area. This length has no defined relationship to any cache line size in the system and software should not make any assumptions to that effect. Based on the size provided by the CPUID instruction, the OS/software would either dynamically allocate structures with appropriate padding or if these were statically allocated and the allocation would cause performance issues with the size, choose to not use Monitor-MWAIT.

While a single length is expected to suffice for single cluster based systems, setting up the data layout correctly for larger systems with multiple clusters would be more complicated. Specifically depending on the coherence mechanism implemented by the chipset in such a system, a single Monitor-line size will not suffice. Typically software will have a set of data variables that it wants to monitor for writes. It is necessary that this set of variables be located in what is referred to as the "monitor triggering area". Furthermore, to eliminate false wake-ups due to other writes to other variables, software will need to add "padding" surrounding this set of variables. This is referred to as the padded area.



A.1.1.1 Waking-up from MWAIT

Multiple events other than a write to the triggering address range can cause a processor that executed the MWAIT instruction to wake up. These include events that would lead to voluntary or involuntary context switches and are listed below.

External interrupts including NMI, INIT, BINIT, MCERR, and interrupts generated by I/O devices.

Faults, Aborts including Machine Check.

Architectural TLB invalidations including writes to CRO, CR3, CR4 and certain MSR writes.

Voluntary transitions due to fast system call and far calls.

A.1.1.2 Avoiding Missing Writes to Monitored Area

Because the software check of the location that is being monitored cannot be done automatically with the execution of the MONITOR instruction, a final check must be made by software after the MONITOR instruction is executed and before the MWAIT instruction is executed. Otherwise, if a write to the location occurs in the window between the time the location is checked by software and the MONITOR instruction is executed, the write will be missed. Refer the Example section under the MWAIT description for more details on how to use the MONITOR and MWAIT instructions.

A.2 MONITOR: Setup Monitor Address

Opcode	Instruction	Description
OF,01,C8	MONITOR EAX, ECX, EDX	Sets up a linear address range to be monitored by hardware and activates the monitor. The address range should be of a write-back memory caching type.

Description: (Similar to the Prescott Processor)

The MONITOR instruction arms the address monitoring hardware using the address specified in EAX. The address range that the monitoring hardware will check for stores can be determined by the CPUID instruction. The monitoring hardware will detect stores to an address within the address range and triggers the monitor hardware when the write is detected. The state of the monitor hardware is used by the MWAIT instruction.

The address range must be in memory of write-back type. Only write-back memory type stores to the monitored address range will trigger the monitoring hardware. If the address range is not in memory of write-back type, the address monitor hardware may not be armed properly.

Note that the address range monitored must be write-back memory type on all cores and/or processors in a CMP and/or multi-processing environment. Otherwise, writes to monitored address range on a particular core that has the range as non-write-back



type will not trigger the monitoring hardware on the core that setup the range using monitor.

The content of EAX is a logical address. By default, the DS segment is used to create a linear address that is then monitored. Segment overrides can be used with the MONITOR instruction. ECX and EDX are used to communicate other information to the MONITOR instruction. ECX specifies optional extensions for the MONITOR instruction. EDX specifies optional hints for the MONITOR instruction and does not change the architectural behavior of the instruction. For the processor, no additional hints or extensions are defined for the Monitor instruction. Specifying undefined hints in EDX are ignored by the processor, whereas specifying undefined extensions in ECX will raise a general protection fault exception on the execution of the MONITOR instruction.

The MONITOR instruction is ordered as a load operation with respect to other memory transactions.

The MONITOR instruction is subject to all permission checking and faults associated with a byte load. Like a load, the MONITOR instruction sets the A-bit but not the D-bit in the page tables.

The MONITOR CPUID feature flag (bit 3 of ECX when CPUID is executed with EAX=1) indicates that a processor supports this instruction. When this flag is set, the unconditional execution of Monitor is supported at privilege levels 0 and conditional execution at privilege levels 1 through 3. Non ring-0 software should test for the appropriate support for these instructions by attempting to execute them prior to their unconditional execution. If the instructions are not supported, they will generate invalid Opcode exceptions. The operating system or IA FW may disable this instruction through IA32_MISC_ENABLES MSR; disabling the instruction clears the CPUID feature flag and causes execution of the MONITOR instruction at any ring-level to generate an invalid Opcode exception (#UD).

Operation

The MONITOR instruction sets up an address range for the monitor hardware using the content of EAX as a logical address and puts the monitor hardware in armed state. The memory address range should be within memory of write-back type. A store to the address range will trigger the monitor hardware. The content of ECX and EDX are used to communicate other information to the MONITOR instruction.

Example: See MWAIT instruction.

Exceptions: None.

Numeric Exceptions: None.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
#GP(0)	ECX specifies extensions.
#SS(0)	For an illegal address in the SS segment.



#PF (fault-code)	For a page fault.
#UD	If CPUID feature flag MONITOR is 0 or the instruction is executed at privilege level 1 through 3 when the instruction is not available if LOCK, REP, REPNE/NZ and Operand Size override prefixes are used.

Real Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH. ECX has a value other than 0.
#UD	If CPUID feature flag MONITOR is 0 or the instruction is executed at privilege level 1 through 3 when the instruction is not available. If LOCK, REP, REPNE/NZ and Operand Size override prefixes are used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode

#PF(fault-code) for a page fault

A.3

MWAIT: Monitor Wait

Opcode	Instruction	Description
OF,01,C9	MWAIT EAX, ECX	A hint that allows the processor to stop instruction execution and enter an implementation-dependent optimized state until the occurrence of class of events.

Description

The MWAIT instruction is designed to operate with the MONITOR instruction to allow a processor to signal an address on which to wait (MONITOR) and an instruction that causes the wait operation to commence (MWAIT). The MWAIT instruction is a hint to the processor that it can choose to enter an implementation-dependent state while waiting for an event or for a store to the address range armed by the preceding MONITOR instruction in program flow.

For example, a HT-capable processor may have one thread enter a low power state while the other thread executes at full speed.

A store to the address range armed by the monitor instruction; an interrupt, NMI, SMI, a debug exception, machine check exception, the BINIT# signal, the INIT# signal, or the RESET# signal will exit the optimized state. Note that an interrupt will cause the processor to exit the optimized state only if the state was entered with interrupts enabled, unless MWAIT hint ECX bit 0 is set to '1b'. If a store to the address range caused the processor to exit then execution will resume at the instruction following the MWAIT instruction. If an interrupt (including NMI) caused the processor to exit the optimized state, the processor will exit the optimized state and handle the interrupt. If an SMI caused the processor to exit the optimized state, execution will



resume at the instruction following the MWAIT after handling of the SMI. Unlike the HALT instruction, the MWAIT instruction does not support a restart at the MWAIT instruction. There may also be other implementation-dependent events or time-outs that may take the processor out of the optimized state and resume execution at the instruction following the MWAIT.

If the preceding MONITOR instruction did not successfully arm an address range or if the MONITOR instruction has not been executed prior to executing MWAIT, then the processor will not enter the optimized state. Execution will resume at the instruction following the MWAIT.

The MWAIT instruction can be executed at any privilege level. The MONITOR CPUID feature flag (bit 3 of ECX when CPUID is executed with EAX=1) indicates that a processor supports this instruction. When this flag is set, the unconditional execution of MWAIT is supported at privilege levels 0 and conditional execution at privilege levels 1 through 3. Non ring-0 software should test for the appropriate support for these instructions by attempting to execute them prior to their unconditional execution. If the instructions are not supported they will generate an invalid Opcode exception. The operating system or IA FW may disable this instruction through IA32_MISC_ENABLES MSR; disabling the instruction clears the CPUID feature flag and causes execution of the MWAIT instruction to generate an invalid Opcode exception (#UD).

In this processor as well as future processor designs, EAX and ECX will be used to communicate information to the MWAIT instruction, such as the kind of optimized state the processor should enter. ECX specifies optional extensions for the MWAIT instruction. EAX may contain hints such as the preferred optimized state the processor should enter. (For the processor, all non-zero values for EAX and ECX are reserved.) A given implementation may choose to ignore the hint and continue executing the next instruction. Future processor implementations may implement several optimized "waiting" states and will select among those states based on the hint argument.

MWAIT Extensions Register (ECX)

Bits	Description
0	Treat Interrupt as break-event, even when interrupts are disabled (EFLAGS.IF=0)
31:1	Reserved = 0

MWAT Hints Register (EAX)

Bits	Description
3:0	Sub C-state within ACPI defined C-states. For C4, When bit 3 is set, disables L2 Shrink for this instruction even if Dynamic L2 shrink is enabled globally by setting PMG_CST_CONFIG_CONTROL MSR bit 3.



7:4	Target C-state Value of 0 means C1; 1 means C2 and so on Value of 01111b means C0
-----	---

Software should use CPUID (EAX=5) leaf to find out if the processor supports the enumeration of Monitor/MWAIT extensions. If bit 0 of ECX is set, the processor supports Monitor/MWAIT extensions.

For CPUID (EAX=5), if bit 1 of ECX is set, the processor supports using interrupt as break-event to MWAIT, even when the interrupts are disabled. This feature can be used to time the C-state residency while making sure that after coming out of MWAIT, the software gets a chance to timestamp before an ISR is potentially executed. Software can set bit 0 in MWAIT Extensions register (ECX) when issuing MWAIT to get this functionality.

For CPUID (EAX=5), various fields in EDX indicate if the processor supports using Monitor/MWAIT to go to various ACPI defined C-states. A processor may support more than 1 C-state of a given ACPI defined C-state type. Software indicates the desired C-state in the MWAIT Hints register (EAX). The processor will never go to any C-state and sub C-state deeper than what is indicated by the software. When CPL>0, for the C-state requests in the hints register ECX, the processor will at most enter C1 state.

Example

The MONITOR and MWAIT instructions must be coded in the same loop because execution of the MWAIT instruction will trigger the monitor hardware. It is not possible to execute MONITOR once and then execute MWAIT in a loop. Setting up MONITOR without executing MWAIT has no adverse effects.

Typically the MONITOR /MWAIT pair is used in a sequence depicted below. Note that the MWAIT Hints and Extensions passed may cause the processor to enter a specified C state.

```

Segment:EAX = Logical Address(Trigger)
ECX = EDX = 0      // Monitor Hints, should be 0 for the processor
If (!trigger_store_happened) {
  MONITOR EAX, ECX, EDX
  If (!trigger_store_happened) {
    EAX = <MWAIT Hints>
    ECX = <MWAIT Extensions>
    MWAIT EAX, ECX
  }
}

```

The above code sequence makes sure that a triggering store does not happen between the first check of the trigger and the execution of the monitor instruction. Without the second check that triggering store would go unnoticed. Typical usage of MONITOR and MWAIT would have the above code sequence within a loop.

Exceptions:

- None

Numeric Exceptions:



- None

Protected Mode Exceptions:

- #GP(0) for any unimplemented extensions specified in extensions register ECX
- #UD if CPUID feature flag MONITOR is 0
- #UD if LOCK prefix is used.

Real Address Mode Exceptions:

- #GP(0) for any unimplemented extensions specified in extensions register ECX
- #UD if CPUID feature flag MONITOR is 0
- #UD if LOCK prefix is used.

Virtual 8086 Mode Exceptions:

- Same exceptions as in Real Address Mode.
- Avoiding Missing Writes to Monitored Area
- MONITOR: Setup Monitor Address

§



Appendix B Windows* 8.1 Micro-PEP(μPEP) ASL Support

B.1 ASL Requirements

Data driven from FW via uPEP _DSM method with following UUID:

`c4eb40a0-6cd2-11e2-bcf0-0800200c9a66`

- uPEP driver will interrogate _DSM during initialization to:
 - Create internal device constraint structures to correctly respond to OSPM callbacks
 - Perform all necessary functions to insure that PO power up callback can be executed within the constrained crashdump environment (non-paged code, pre-mapped IO, and so on.)
- For Device Constraints: ASL needs to enumerate the following:
 - ACPI device descriptor
 - Device enabled/disabled indicator
 - Constraints package (constraint per Low Power Idle state as described in LPIT)
- For Crashdump: ASL needs to enumerate the following:
 - Which device(s) need to support crashdump
 - Descriptor of the uPEP accessible power control(s)
 - Information on how to access the power control(s)
 - Timeout wait period (if needed) before returning to PO or executing subsequent commands.

B.2 Referenced Documents

The ACPI Low Power Idle Table (LPIT) is an ACPI 6.0 proposal currently under review. Refer Intel ASL reference code for the most current LPIT and uPEP ASL tables.

B.3 Device Constraints Enumeration

For each platform device, specify the Platform Idle State constraints in terms of minimum D state or OEM-specific State. The ACPI name “DEVY” is not prescriptive but must match what is returned through _DSM function 1 below. The Device enabled/disabled parameter enables BIOS to enumerate all possible devices and dynamically enable/disable as needed during BIOS initialization.

```
Name(DEVY, Package()
{
    //
    // 1: ACPI Device Descriptor: Fully Qualified namestring
```



```
// 2: Enabled/Disabled Field

//      0 = This device is disabled and applies no constraints
//      >0 = This device is enabled and applies constraints

// 3: Constraint Package: entry per LPI state in LPIT, based on ACPI
_LPD proposal

//      a. Constraint package revision: currently 0

//      b. One or more constraint Tuple packages, consisting of:
//          i. Associated LPI State UID

//          ID == 0xFF: same constraints apply to all states in
LPIT

//          ii. minimum Dx state as pre-condition

//          iii. (optional) OEM specific - OEM may provide an
additional encoding

//          which further defines the D-state Constraint

//          0x0-0x7F - Reserved

//          0x80-0xFF - OEM defined

//

Package() {"\\_PR.CPU0", 0x1, Package() {0, Package(){0xFF, 0}}},  
Package() {"\\_PR.CPU1", 0x1, Package() {0, Package(){0xFF, 0}}},  
Package() {"\\_PR.CPU2", 0x1, Package() {0, Package(){0xFF, 0}}},  
Package() {"\\_PR.CPU3", 0x1, Package() {0, Package(){0xFF, 0}}},  
Package() {"\\_SB.I2C1", 0x1, Package(){0, Package(){0xFF,3}}},  
Package() {"\\_SB.I2C2", 0x1, Package(){0, Package(){0xFF,3}}},  
Package() {"\\_SB.I2C3", 0x1, Package(){0, Package(){0xFF,3}}},  
Package() {"\\_SB.I2C4", 0x1, Package(){0, Package(){0xFF,3}}},  
Package() {"\\_SB.I2C5", 0x1, Package(){0, Package(){0xFF,3}}},  
Package() {"\\_SB.I2C6", 0x1, Package(){0, Package(){0xFF,3}}},  
Package() {"\\_SB.I2C7", 0x1, Package(){0, Package(){0xFF,3}}},  
Package() {"\\_SB.PCI0.GFX0", 0x1, Package(){0, Package(){0xFF,3}}},  
Package() {"\\_SB.PCI0.SEC0", 0x1, Package(){0, Package(){0xFF,3}}},  
Package() {"\\_SB.PCI0.XHCI", 0x1, Package(){0, Package(){0xFF,3}}},
```



```

    Package(){"\\_SB.PCI0.GFX0.ISP0", 0x1, Package(){0,
    Package(){0xFF,3}}},

    Package(){"\\_SB.LPEA", 0x1, Package(){0, Package(){0x0,3},
    Package(){0x1,0}, Package(){0x2,3}, Package(){0x3,3}}},

    Package(){"\\_SB.SDHA", 0x1, Package(){0, Package(){0xFF,3}}},

    Package(){"\\_SB.SDHB", 0x1, Package(){0, Package(){0xFF,3}}},

    Package() {"\\_SB.SDHC", 0x1, Package(){0, Package(){0xFF,3}}},

    Package() {"\\_SB.SPI1", 0x1, Package(){0, Package(){0xFF,3}}},

    Package() {"\\_SB.URT1", 0x1, Package(){0, Package(){0xFF,3}}},

    Package() {"\\_SB.URT2", 0x1, Package(){0, Package(){0xFF,3}}}

})

```

BIOS must insure that all enumerated devices in this _DSM function represent devices present on the system with correctly formed ACPI device descriptors. If an enumerated device can be disabled, for example via BIOS switches, the device must also be disabled within the _DSM function.

If the same Min constraint (either Dx or OEM specific) applies to all available LPI States (as defined in the LPIT), the LPI UID can be set to 0xFF to simplify implementation.

If different minimum constraints applied per different LPI States (as defined in the LPIT), then each LPI state/constraint tuple must be defined uniquely. In this case, the tuples should be defined starting with LPI state 0 and each subsequent tuple will be monotonically increasing to match the LPI State Unique ID requirement. In other words if 3 LPI states exist, and device constraints are D0, D3 and D3 respectively, then the package would look like this:

```

Package() {
    "\\_SB.ADSP",           // ACPI device descriptor for Audio DSP
    0x1,                   // Enable/Disable bit: 1 == Enabled
    Package() {
        "_LPD",           // Constraints Tuple package, based on ACPI
        0,                 // Constraints revision: currently 0
        Package(){0x0, 0},  // LPI state 1 constraint: D0
        Package(){0x1, 3},  // LPI state 2 constraint: D3
        Package(){0x2, 3}   // LPI state 3 constraint: D3
    }
}

```



The Constraints Tuple package is equivalent to the proposed ACPI _LPD object to insure forward compatibility.

An ACPI notification 0x80 to the PEP device, Notify(_SB.PEPD, 0x80), will cause a re-evaluation of the _DSM functions. This would be used in cases where devices are added/ removed from the system, such as a dock event.

B.4 Bugcheck Critical Device Enumeration

The Name "BCCD" is not prescriptive but must match what is returned in _DSM function 2 below.

The following ASL is for example purposes only and may or may not describe an actual implementation.

```
Name(BCCD, Package()
{
    //
    // Bugcheck Critical Device(s)
    //
    // 1: ACPI Device Descriptor: Fully Qualified name string
    //
    // 2: Package of packages: 1 or more specific commands to power up
    // critical device
    //
    // 2a: Package: GAS-structure describing location of PEP accessible
    // power control
    //
    //      Refer ACPI 5.0 spec section 5.2.3.1 for details
    //
    //      a: Address Space ID (0 = System Memory)
    //
    //          NOTE: A GAS Address Space of 0x7F (FFH) indicates remaining
    //          package
    //
    //                  elements are Intel defined
    //
    //      b: Register bit width (32 = DWORD)
    //
    //      c: Register bit offset
    //
    //      d: Access size (3 = DWORD Access)
    //
    //      e: Address (for System Memory = 64-bit physical address)
    //
    // 2b: Package containing:
    //
    //      a: AND mask - not applicable for all Trigger Types
    //
    //      b: Value (bits required to power up the critical device)
```



```

//      c: Trigger Type:
//
//          0 = Read
//
//          1 = Write
//
//          2 = Write followed by Read
//
//          3 = Read Modify Write
//
//          4 = Read Modify Write followed by Read
//
//  2c: Power up delay: Time delay before next operation in uSec
//
//      Package()
{
    "\_SB.SDHA",
    Package()
{
    Package(){ Package() {0, 32, 0, 3, 0xFFFFFFFFFFFFFF}, 0}
    Package(){0xFFFFFFF, 0x0, 0x4}, 0
}
}
}
)

```

If a mechanism to power up controllers/devices cannot be described via the ACPI Generic Address Structure (GAS) like structure (example: UEFI interface or MSR access), the GAS encoding Address Space ID of 0x7F will indicate specific encoding as described in the table below

All crashdump devices must be listed in the BCCD function. Crashdump device register with PoFxRegisterCrashdumpDevice(). PoFx calls PEP, but if no matching device exists in the BCCD list, then uPEP must return STATUS_NOT_SUPPORTED.

If a crashdump device registers, but does not need specific power up actions (for example, does not support D3cold), then the PEP needs to recognize the device and that no actions are needed. For this case, the Package GAS Address Space ID of 0x7F will indicate specific encoding as described in the table below.

Valid GAS Address space IDs for BCCD functions are restricted to the following:

Address Space ID	Description
0	System Memory
1	System I/O



Address Space ID	Description
0x7F	Functional Fixed Hardware (Refer table below)

For System Memory and System I/O types, access sizes of BYTE, WORD, DWORD are supported. QWORD System Memory type is also supported on 64-bit systems. The register Bit Width must match the access size (BYTE access supports 8-bit register only, WORD access supports 16-bit register only, etc.) The register address must fall on a naturally aligned boundary based on access size i.e. DWORD access size supports a 32-bit wide register and address falls on a 4-byte boundary. These restrictions further dictate that Register Bit Offset field must therefore be zero.

The "AND mask" and "Value" package components must match the "Register Bit Width" as described in the associated GAS structure package.

Field	Byte Length	Byte Offset	Values to Indicate MSR Access	Values to indicate "No Action Required"
Address Space ID	1	0	0x7F	0x7F
Register Bit Width	1	1	64 to indicate the MSR bit width	1
Register Bit Offset	1	2	0	0
Access Size	1	3	0	0
Address	8	4	<MSR value>	0

MSR accesses are natively accessed as a QWORD access size.

B.5 _DSM Example

```
//  
// _DSM - Device Specific Method  
//  
// Arg0: UUID Unique function identifier  
// Arg1: Integer Revision Level (currently 0)  
// Arg2: Integer Function Index  
// Arg3: Package Parameters  
//  
Method(_DSM, 0x4, Serialized)  
{  
    // New UUID for built-in uPEP  
    If(LEqual(Arg0, ToUUID("C4EB40A0-6CD2-11E2-BCFD-0800200C9A66")))  
    {  
  
        // Number of fn IDs supported  
        If(LEqual(Arg2, Zero))  
        {  
            Return(Buffer(One)  
            {  
                0x7  
            })  
        }  
    }  
}
```



```
        }
        // LPI device dependencies
        If(LEqual(Arg2, 0x1))
        {
            Return(DEVY)
        }
        // Crashdump device data
        If(LEqual(Arg2, 0x2))
        {
            Store(EM1A, Local0)
            Add(Local0, 0x84, Local0)
            Store(Local0,
Index(DerefOf(Index(DerefOf(Index(DerefOf(Index(DerefOf(Index(BCCD,
Zero, )),
One, )),
Zero, )),
Zero, )),
0x4, ))
            Return(BCCD)
        }
    }

    Return(One)
}
```

§