Skylake/Kaby Lake H/LP Platform Controller Hub (PCH)

BIOS Specification

November 2017

Revision 2.3.4

Intel Confidential

Document Number: 549921

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.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at Intel.com, or from the OEM or retailer.

No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.

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 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 technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM 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 requires a system with an appropriate Intel chipset and a motherboard with an appropriate codec and the necessary drivers installed. System sound quality will vary depending on actual implementation, controller, codec, drivers and speakers. For more information about Intel® HD audio, refer to http://www.intel.com/.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries. See Trademarks on intel.com for full list of Intel trademarks.

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

Copyright © 2014-2016, Intel Corporation. All rights reserved.

Contents

1 Introduction 25
1.1 Terminology 25
1.2 Related Documents 27

2 Intel Platform Controller Hub (PCH) Overview 29
2.1 Platform Controller Hub (PCH) Internal PCI Devices and Functions 31
2.2 Primary to Sideband Bridge (P2SB) 31
2.2.1 SBI Programming 32
2.2.2 MMIO Back to Back Write Restriction and Handling 33
2.2.3 Endpoint Mask 33
2.2.4 SBILOCK 33
2.3 Memory-Mapped Chipset Configuration Space 34
2.3.1 Memory-Mapped Registers Affected by Posted-Write 35
2.4 Platform Controller Hub (PCH) Configuration Register Changes 36
2.5 Platform Controller Hub (PCH) Cycle Decoding 36
2.5.1 DMI Cycle Decoding 37
2.5.2 PSF Cycle Decoding 39
2.5.3 LPC Cycle Decoding 40
2.5.4 LPC Memory Cycle Decoding 40
2.5.5 I/O Port 80h Cycle Decoding 41
2.6 Recommended Access Mechanism 42

3 Flash Programming Considerations 43
3.1 Overview 43
3.2 Boot BIOS and eSPI Strap 43
3.3 Flash Memory Map 44
3.4 Flash Device Enumeration 44
3.4.1 ACPI ASL Definition 45
3.5 SPI Specific Configuration 45
3.5.1 SPI Configuration Registers 45
3.5.2 SPI Read Configuration 45
3.5.3 PCH SPI Initialization Flow 46
3.5.4 Serial Flash Discoverable Parameter (SFDP) 46
3.5.5 SPI Flash Operation Modes 46
3.6 Flash Security Recommendation 47
3.7 BIOS Region SMM Protection Enabling 49
3.8 PCH Soft Strap Access 49

4 Reset Control 50
4.1 Reset Control Overview 50
4.2 Hard Reset Control 50
4.3 CPU Only Reset Control 50
4.3.1 Additional Concern of CPU Only Reset 50
4.4 Special Cycle Reset 51
4.5 Global Reset 51
4.6 SLP_S3#, SLP_S4#, SLP_SUS#, and SLP_A# Stretch Requirements 51
4.7 Processor Configuration Change Reset Requirement 51

5 Basic BIOS Requirements 52
5.1 Initializing Platform Controller Hub SBREG Base Address 52

5.1.1 Hiding P2SB 52
5.2 Initializing the I/O Base Registers 52
5.3 Hiding and Function Disable Platform Controller Hub (PCH) Internal Devices 53
5.3.1 Fuse Disabled, Static Disabled, Non-Static Function Disable Device Handling 54
5.4 Platform Management Timers 54
5.4.1 Periodic SMI timer 55
5.4.2 Software SMI Timer 55
5.5 PCI/PnP Enumeration 55
5.6 ACPI Support 55
5.7 Reporting IO/Memory Resources to OS 56
5.8 Cache Line Size Clarification 57
5.9 Intel Reserved Memory Address Space 57
5.9.1 Special PCH BIOS Reserved Memory Allocation 57
5.10 R/WO Registers 59
5.11 Platform Controller Hub (PCH) Boot Checklist 60
5.12 Intel® Stable Image Platform Program (SIPP) 60
5.13 BIOS Guide on Using RTC RAM 61
5.14 Additional Programming Steps 61
5.14.1 Processor Set Strap Message 61
5.14.2 EFI Tick Rate 61
5.14.3 PSF Register Access 61
5.14.4 PSF EOI Multicast Configuration 61
5.14.5 VTd Support Requirement 62
5.14.6 Additional Power Management Configuration 63
5.15 Legacy IO low latency mode support 63

6 Interrupt Support on Platform Controller Hub (PCH) 64
6.1 PCI IRQ Routing on Platform Controller Hub 64
6.1.1 PCI* Interrupt to IRQ Router 64
6.1.2 Interrupt Routing for Platform Controller Hub (PCH) Internal PCI Devices 66
6.1.3 Interrupt Routing for PCI Express* Root Ports 68
6.1.4 Interrupt Routing for DMI Assert_INTx Messages 69
6.1.5 Interrupt Routing for PCI Expansion Slots 70
6.1.6 Reporting Interrupt Routing to OS 71
6.2 ACPI Device IRQ Routing on Platform Controller Hub 73
6.2.1 Interrupt Routing for Internal ACPI Devices 73
6.2.2 Interrupt Routing for External Devices through GPI 74
6.3 Serial IRQs 75
6.4 PIRQ Enable/Disable Handling by Platform Controller Hub (PCH) 75
6.5 NMI Handling in Platform Controller Hub (PCH) 76
6.5.1 PCH Settings to Generate NMI 76
6.5.2 NMI Handling in an OS Environment 76
6.5.3 Steps for Handling NMI in Platform Controller Hub (PCH) Systems 77
6.5.4 Routing NMI to SMI 81
6.6 SMI Handling in Platform Controller Hub (PCH) 82
6.6.1 PCH Settings to Generate SMI 82
6.6.2 SMI LOCK Bit 82
6.6.3 Setting the EOS Bit 82
6.6.4 SMI Status Bits 82
6.7 Processor Interrupt Support 82
6.7.1 Dual/Quad Processor Considerations 83

6.7.2 Processor Interrupt Delivery Mechanism 83

7 Direct Media Interface (DMI) and On-Package Direct Media Interface (OP-DMI) 85
7.1 DMI Link Speed and Training 85
7.1.1 Gen 2/3 Link Speed Configuration 86
7.2 Platform Controller Hub (PCH) DMI/OPIO Configuration 86
7.2.1 DMI Virtual Channel 86
7.2.2 DMI Link Active State Power Management (ASPM) 87
7.2.3 PCH Display Port 87
7.2.4 Additional PCH DMI and OP-DMI Programming Steps 87

8 PCI Express* Support 92
8.1 PCI Express* Configuration Space Base Address 92
8.1.1 Register Access Using Capabilities List 92
8.1.2 Device/Port Type Field of PCI Express* Devices 93
8.2 Configuration of PCI Express* Root Ports 93
8.2.1 PCIe* Function Disable Requirement 94
8.2.2 Port Width Configuration 96
8.2.3 Initialize “Slot Implemented” for Root Ports 97
8.2.4 Initialize “Physical Slot Number” for Root Ports 97
8.2.5 Initialize “Slot Power Limit” for Root Ports 97
8.2.6 SCI/SMI Generation 97
8.2.7 Build Capability Linked List and Extended PCIe* Capability Linked List 98
8.2.8 Maximum Payload Size 99
8.2.9 PCIe* Relaxed Order 99
8.2.10 Completion Retry Status Replay Enable 99
8.2.11 Peer Memory Access 100
8.3 ASPM on DMI and the PCI Express* Root Ports 100
8.3.1 Calculation of L-State Exit Latency 102
8.4 SERR# Generation 103
8.5 PCI Firmware Spec 3.0 Support 104
8.6 ACPI Table and Methods for PCI Express* Support 104
8.6.1 _OSC() Method 104
8.7 PCI Express* PME BIOS Support 104
8.7.1 Native PME Software Model 105
8.7.2 Legacy PME Software Model 105
8.7.3 BIOS Enabling of Platform Controller Hub (PCH) PCI Express* PME SCI Generation 105
8.7.4 Handling PCI Express* PME SCI Event 107
8.7.5 Transition from Legacy to Native PME Software Model 108
8.7.6 WAKE# Support 109
8.8 PCI Express* Hot Plug BIOS Support 109
8.8.1 Native Hot Plug Software Model 109
8.8.2 Legacy Hot Plug Software Model 110
8.8.3 Transition from Legacy to Native Hot Plug Software Model 114
8.8.4 PCIe* Device Hot Plug Enabling 115
8.9 ExpressCard* Support 115
8.10 PCI Bus Emulation and Port 80 Decode Support 117
8.11 Sample ASL Code for PME/ Hot Plug Support 117
8.12 Non-Common Clock with SSC 120
8.13 IOSF Port Configuration and Grant Count Programming 120
8.14 PCI Express* Gen 3 Support 122
8.14.1 Static Equalization Method 122

8.14.2 Software Equalization Method 123
8.15 Additional PCI Express* Programming Steps 125
8.15.1 Power Optimizer Configuration 127

9 Intel® High Definition Audio (Intel® HD Audio) Controller 130
9.1 Intel® HD Audio Codec Initialization 130
9.1.1 Intel® HD Audio Codec Architecture Introduction 130
9.1.2 Codec Verb Table 132
9.1.3 Codec Initialization Programming Sequence 133
9.1.4 Intel® HD Audio Codec Initialization on S3 Resume 139
9.2 Intel® HD Audio Controller Configuration 140
9.3 Intel® HD Audio PME Event 140
9.4 Intel® HD Audio Isochronous (“Isoch”) Support on VC1 140
9.4.1 DMI VC1 Configuration 140
9.4.2 Intel® HD Audio VC1 Configuration 141
9.5 Intel iDisplay Audio Link 141
9.5.1 Intel iDisplay Audio Verb Table 141
9.6 Additional Intel® HD Audio Programming Steps 143
9.6.1 PCI Mode 143
9.6.2 Function Disabling Steps 143
9.6.3 Misc Initialization Steps 143
9.6.4 Static Frequency Switching 144
9.6.5 Registers Lock Down and Related Field 145
9.6.6 Power Management and Clock Gating Requirement 146

10 Gigabit Ethernet Controller 147
10.1 GbE Controller Support 147
10.2 Enabling / Disabling the Internal GbE Controller 147
10.2.1 Enable the Internal GbE Controller 147
10.2.2 Disable the Internal GbE Controller 147
10.3 Accessing LAN MAC address Recommendation 148
10.4 Wake-On-LAN (WOL) Implementation 149
10.5 Additional GbE Controller Configurations for WOL Support 149
10.6 Additional GbE-based Wake Events 151
10.7 Retrieve Lan Phy revision 151
10.8 Power Optimizer Configuration 152
10.8.1 LTR Programming 152
10.8.2 Clock and Lock Control Configuration 152

11 SMBus 153
11.1 SMBus Usage Model 153
11.2 Reading a Byte of SMBus EEPROM Data 153
11.3 Block Mode 154
11.3.1 Block Writes 154
11.3.2 Block Reads 154
11.4 BIOS / Driver Hardware Semaphore 155
11.5 Wake Up from SMBus 155
11.6 SMBus Interaction with TCO 155
11.7 SMBus 2.0 Support 155
11.8 SMBus Block Transfers and TCO 156
11.9 Support for Handling Incomplete SMBus Transactions 156
11.10 SMBus System BIOS Responsibilities 157
11.11 SMI/OS Driver Resource Arbitration 158

11.11.1 SMBus Host Controller Ownership 158
11.11.2 SMBus Host Controller Interrupt Ownership 160
11.12 SMBus Handling for System Resets 161
11.13 Additional Programming Requirement 163

12 Extensible Device Controller Interface (xDCI) 164
12.1 xDCI Enabling and Disabling 164
12.2 PCI/ACPI Modes 164
12.2.1 PCI Mode of Operation 165
12.2.2 PCI Mode Interrupt Pin Mapping 165
12.3 Power Management Requirement 165

13 XHCI Support 166
13.1 BIOS Core Requirements 166
13.2 Legacy USB Support 166
13.2.1 Legacy USB Support and Dual Processor Issues 166
13.2.2 Handoff of USB Controller to the OS 167
13.2.3 Implementing Legacy USB Keyboard Support 167
13.2.4 USB Legacy Operation Requirements 167
13.3 Software IRQ Generation in Platform Controller Hub (PCH) 169
13.4 xHCI Controller Initialization 170
13.4.1 xHCI Controller Setup 171
13.4.2 Super Speed Port Enable 172
13.4.3 xHCI SSIC Interface Configuration 172
13.4.4 Port Disable Override 175
13.4.5 Hiding/Disabling xHCI Controller 175
13.4.6 Locking xHCI Register Settings 175
13.4.7 xHCI Legacy SMI Enabling 176
13.4.8 Host Controller Reset or WPR 176
13.5 D3 enabling 176
13.6 Debug Capabilities Support 177
13.7 DCI Support 178

14 Integrated SATA Controllers 179
14.1 SATA Controller Initialization 179
14.1.1 Setting the SATA Controller Mode 180
14.1.2 Enable Ports 181
14.1.3 Initialize Registers in AHCI Memory-Mapped Space 182
14.1.4 RAID Mode Initialization 183
14.1.5 Power Optimizer Configuration 186
14.1.6 Additional Programming Requirements during SATA Initialization 187
14.1.7 External SATA Programming 188
14.1.8 Compliance to Industry Specifications 188
14.2 DevSleep Enabling 189
14.2.1 DevSleep Warm Reset Flow Handling 190
14.3 SATA Controller Disabling 191

15 High Performance Event Timer (HPET) Support 193
15.1 High Performance Event Timer Description Table (HPET) 193
15.2 HPET ASL CODE 194

16 GPIO Handling 195
16.1 ACPI Compliant GPIO 195
16.1.1 1-Tier GPEs 195

16.1.2 2-Tier GPEs....196
16.1.3 Example GPE configuration....196
16.1.4 ACPI GPE handling....197
16.2 GPIO reset configuration....197
16.3 Serial GPIO Feature....199
16.4 GPIO Programming Steps....199
16.4.1 GPIO Pad in Native Mode....199
16.4.2 GPIO Pad in ACPI....200
16.4.3 GPIO interrupts....201
16.4.4 Termination....203
16.4.5 Not connected pad recommended settings....204
16.4.6 Pad Mode Enabling for PWM Mode....204
16.5 GPIO Pad Ownership....204
16.6 GPIO Registers Lockdown....205
16.7 GPIO VISA/PTI Pin Support....205
16.8 GPIO Pad Special Handling....206
16.8.1 PCH-LP GPP_A_12....206
16.8.2 General Purpose SPIs' Chip Select Pads....206
16.8.3 SATA DevSlp....206
16.9 GPIO Pin Isolation During Sx....206
16.10 GPIO Device ACPI Requirement....207

17 Thermal Management Support....208
17.1 Thermal Management Features Overview....208
17.2 Thermal Subsystem Device Initialization....208
17.3 Thermal Sensor Initialization....209
17.3.1 Initializing Thermal Sensors....209
17.4 Thermal Reporting Requirements....211
17.4.1 Thermal Trip Pin....212
17.5 Thermal Throttling....212
17.5.1 Memory Bandwidth Throttling....214

18 Integrated Clock Support....215
18.1 ICC Overview....215
18.1.1 Full Clock Integration Mode Overview....215
18.2 BIOS Requirement....215
18.2.1 ICC MEI Messages....215
18.2.2 ICC MEI Messages Definition....220
18.2.3 MEI MKHI Messages....236
18.2.4 Full Clock Integration Mode....238
18.3 Clock Gating....240
18.4 ICC Profile Selection and Verification....241
18.5 Watchdog Support....241
18.5.1 WDT Support for Basic ICC configuration....242
18.5.2 WDT Support for Extended ICC Configuration....242
18.6 ICC Control Library on Skylake/Kaby Lake....242

19 Power Management....243
19.1 Initialization....243
19.2 Handling Status Registers....243
19.3 Power Button Override and AfterG3_EN Bit Usage....243
19.3.1 Intel® ME Enabled System after G3 Exit Behavior....244
19.4 Power Failure Considerations....244

19.5 Additional Power Management Programming 245
19.6 Generate SMI on SLP_EN Write 247
19.7 PCI Express* Bus PME# Signal Based Wakeup....247
19.8 RTC Considerations 247
19.8.1 RTC Resets 247
19.8.2 UIP Considerations 247
19.8.3 Alternate Access Mode 248
19.9 Wake Events in the Intel PCH 248
19.10 Deep S3 / S4 / S5 Power Policies 249
19.11 Suspend Handler Considerations....250
19.11.1 S3 (Suspend to RAM) Steps 250
19.11.2 Restoring Of PCI Registers after a Resume from STR....251
19.12 Power Optimizer Considerations (MB Only)....251
19.13 PCI Express Output Clock Enable/Disable Control....251
19.14 LPC Power Saving Consideration 252
19.15 SPI/eSPI Power Management Programming....252
19.16 PSTH Power Management Programming....252

20 TCO Support 254
20.1 Watchdog Operation....254
20.1.1 Second Timeout STS 254
20.2 Checking TCO Timer after Reset 254
20.3 Intruder Detect....255
20.3.1 TCO Message 255
20.4 Century Rollover 255
20.4.1 Additional SMI handling of NEW CENTURY of Skylake/Kaby Lake ..255

21 Legacy Free Systems 257
21.1 Legacy Free Systems Handling by BIOS....257
21.2 Virtual Legacy Wire (VLW) Messages 257
21.3 Reset Mechanism 257
21.3.1 Fixed ACPI Description Table Boot Architecture and Flags Description 258
21.4 Debug Port Table 258
21.5 Summary of Legacy-Free Handling in ACPI....258
21.6 Port 61h Bit 4 Non-toggling Implication 259
21.7 Legacy DMA Removal Implication....260

22 Intel® Serial I/O Bus 261
22.1 Intel® Serial I/O Initialization Flow 262
22.2 Disabling Intel® Serial I/O Controllers....263
22.2.1 Static Power Gating and Re-enabling of Intel® Serial I/O Devices .264
22.3 Intel® Serial I/O BAR Resource Allocation 264
22.4 UART OS Debug Port Support 264
22.5 Intel® Serial I/O SPI Additional Programming Requirement 266
22.6 Intel® Serial I/O PCI Mode Initialization 266
22.7 Intel® Serial I/O ACPI Mode Initialization 268
22.7.1 Intel® Serial I/O ACPI Mode Requirements....269
22.7.2 Hidden Mode....270
22.7.3 VTd Support....270
22.8 Intel® Serial I/O Power Management Programming 270
22.8.1 Power States....270
22.8.2 Clock/PM Initialization 271

23 Intel® Smart Sound Technology (Intel® SST) 272
23.1 Intel® Smart Sound Technology Initialization Flow 272
23.2 Intel® Smart Sound Technology Enable/Disable 272
23.3 Additional Programming Steps 273
23.4 ACPI Requirements 273
23.4.1 Non-HD-A Link Descriptor Table 273
23.4.2 Feature Support 276
23.4.3 3rd Party IP 276
23.5 Intel Proprietary Wake Support 278

24 Intel® Rapid Storage Technology (Intel® RST) for PCIe* Storage 279
24.1 PCIe* Storage Overview 279
24.2 Remapping and Cycle Router 279
24.3 Discovery and Initialization 280
24.3.1 PCH SKU Detection 280
24.3.2 Initialization 281
24.3.3 Fast Boot/Sx Resume Handling 286
24.3.4 Manual Port Selection 286
24.3.5 PCIe* Capabilities Enabling 287
24.3.6 ACPI Requirement 288
24.3.7 SATA Controller Dependency 288
24.3.8 RTD3 Consideration 288
24.3.9 SMM Consideration 288
24.3.10 Limitation 288
24.4 Platform Handling 289
24.5 RST One Click Optane Enable 289
24.5.1 Purpose 289
24.5.2 Mode of operation 289
24.5.3 Variable definition 289
24.5.4 Platform prerequisites 290
24.5.5 Synchronization with BIOS setup 290
24.5.6 Security considerations 290

25 Intel® Sensor Hub (ISH) 291
25.1 ISH Disable/Enable Handling 291
25.2 PCI/ACPI Modes 291
25.2.1 PCI Mode of Operation 291
25.2.2 PCI Mode Interrupt Pin Mapping 291
25.3 Power Management Programming 291
25.4 Initialization Requirement 292
25.5 Intel Proprietary Wake Support 292
25.6 MEI MKHI Messages 292
25.6.1 PDT Unlock Data Message 292

26 Camera I/O Controller (LP) 294
26.1 CIO2 Controller Enabling 294
26.2 CIO2 Controller Disabling 294
26.3 Device Mode 294
26.4 Additional Initialization Programming Steps 294
26.5 ACPI Requirements 295

27 eSPI 296
27.1 eSPI Enabling 296
27.2 eSPI Device Interrupt Support 296

27.3 Tunneled Slave Access Programming 296
27.3.1 Get_Configuration 296
27.3.2 Set_Configuration 297
27.3.3 Get_Status 297
27.3.4 In-band Reset 297
27.4 Initialization Requirement 298

28 Storage and Communication Subsystem 299
28.1 SCS Initialization Flow 299
28.2 SCS Devices Enable/Disable Handling 299
28.2.1 D30F7 Disable 299
28.2.2 SDIO / SD Card 299
28.2.3 SD Card Specific Programming 300
28.2.4 eMMC 300
28.3 SNOOP / Non-SNOOP Programming 300
28.4 Power Management and Clock Gating Programming 300
28.5 SD DLL Programming 300
28.6 eMMC DLL Programming 301
28.6.1 eMMC HS400 Software Training Sequence 301
28.6.2 High Level Flow of eMMC HS400 Training 302
28.6.3 HS400 Mode Selection 302
28.6.4 Rx Data Path Training 302
28.6.5 Tx Data Path Training 303
28.6.6 Fast Path and S3 Path Consideration 304
28.7 PCI/ACPI Modes 304
28.7.1 PCI Mode of Operation 304
28.7.2 PCI Mode Interrupt Pin Mapping 304
28.7.3 ACPI Mode of Operation 304
28.7.4 ACPI Mode Interrupt Assignment 305
28.8 SCS ACPI Requirements 305
28.8.1 Example of ASL code for eMMC/SDIO/SDXc Connected Device 305

29 HSIO and FIA 309
29.1 FIA Lane Ownership and Handling 309
29.1.1 CLKREQ to Root Port Mapping 310
29.1.2 Runtime Trigger 310
29.2 MEI MKHI Messages 311
29.3 Power Management Programming 312

30 Intel® Trace Hub (Intel® TH) 314
30.1 Intel® TH Initialization 315
30.1.1 Further Initialization Steps for Post Memory 316
30.2 BIOS Debug Trace Usage 317

31 Lewisburg PCH 319
31.1 Introduction 319
31.2 Intel Platform Controller Hub (PCH) Overview 319
31.2.1 To 2.2 Primary to Sideband Bridge (P2SB) 320
31.2.2 To 2.5.3 LPC Cycle Decoding 320
31.2.3 SVID and SID of PCH Devices 321
31.3 Flash Programming Considerations 321
31.4 Reset Control 321
31.4.1 Demoted Warm Reset 321
31.4.2 PEI Phase related flow 323

31.4.3 PCIe Root Port related flow 323
31.4.4 Power Management related flow 324
31.5 Basic BIOS Requirements....324
31.6 Interrupt Support on Platform Controller Hub (PCH) 324
31.7 Direct Media Interface (DMI) and On-Package Direct Media Interface (OP-DMI) 325
31.8 PCI Express Support 325
31.8.1 To 8.15 Additional PCI Express* Programming Steps 325
31.8.2 To 8.2.1.1 PCIe* Function Disable Handling 325
31.8.3 To 8.15.1 Power Optimizer Configuration....325
31.8.4 End of PEI 326
31.8.5 DWR flow 326
31.8.6 Maximum Payload Size....326
31.9 Intel® High Definition Audio Controller....326
31.9.1 To 9.6.6 Power Management and Clock Gating Requirement 326
31.10 Gigabit Ethernet Controller....326
31.10.1 To 10.8.1 LTR Programming....326
31.10.2 DXE phase flow 327
31.11 SMBus....327
31.11.1 Disabling SMBus....327
31.11.2 Post-mem init 327
31.12 Extensible Device Controller Interface (xDCI) 327
31.12.1 Pre-mem 327
31.12.2 To 12.1 xDCI Enabling and Disabling....327
31.13 XHCI Support 328
31.13.1 To 13.4.1 xHCI Controller Setup 328
31.13.2 To 13.4.2. Super Speed Port Enable....329
31.13.3 To 13.7 DCI Support 329
31.13.4 Locking xHCI Register Settings....329
31.14 Integrated SATA Controllers....329
31.14.1 To 14.1 SATA Controller Initialization 330
31.14.2 To 14.1.3 Initialize Registers in AHCI Memory-Mapped Space 330
31.14.3 To 14.1.5 Power Optimizer Configuration....330
31.14.4 To 14.1.6 Additional Programming Requirements during SATA Initialization....330
31.14.5 To 14.3 SATA Controller Disabling 331
31.14.6 Additionally for LBG: 331
31.15 High Performance Event Timer (HPET) Support....331
31.16 GPIO Handling....331
31.16.1 To 16.4 GPIO Programming Steps....331
31.16.2 To 16.8 GPIO Pad Special Handling....331
31.17 Thermal Management Support....332
31.17.1 To 17.5 Thermal Throttling 332
31.18 Integrated Clock Support 332
31.19 Power Management....332
31.19.1 To 19.5 Additional Power Management Programming 332
31.19.2 To 19.15 SPI/eSPI Power Management Programming....334
31.20 TCO Support 334
31.20.1 To 20.4.1. Additional SMI handling of NEW CENTURY of Skylake/Kaby Lake 334
31.21 Legacy Free Systems....334
31.22 Intel Serial I/O Bus 334
31.22.1 To 22.2 Disabling Intel® Serial I/O Controllers....334

31.22.2 End of PEI 334
31.23 Intel® Smart Sound Technology 335
31.24 Intel® RST for PCIe Storage 335
31.24.1 To 24.3.2. Initialization 335
31.25 Intel Sensor Hub 335
31.25.1 To 25.1 ISH Disable/Enable Handling 335
31.26 Camera I/O Controller (LP) 336
31.26.1 To 26.2 CIO2 Controller Disabling 336
31.27 eSPI 336
31.27.1 To 2.5.1.2 LPC Generic I/O Range Decoding 336
31.27.2 To 2.5.1.3 LPC Memory Range Decoding 336
31.27.3 To 2.5.1.6 LPC IO Enables Decoding 337
31.28 Storage and Communication Subsystem 337
31.29 HSIO and FIA 337
31.29.1 To 29 HSIO and FIA 337
31.30 Intel® Trace Hub 338
31.30.1 To 30.1 Intel® TH Initialization 339
31.31 Management Component Transport Protocol (MCTP) 339
31.32 Reliability, availability and serviceability (RAS) 340
31.32.1 PEI Phase flow 340
31.33 Additional Lewisburg Devices 340
31.33.1 sSATA Controller/Ports 340
31.33.2 MROM 0 and MROM 1 341
31.33.3 Acceleration Complex. 343
31.33.4 Innovation Engine 344
§ § 344

Figures

Figure 2-1: Skylake/ Kaby Lake PCH-LP and PCH-H Block Diagram 31
Figure 2-2: Platform Controller Hub (PCH) Cycle Decoding 37
Figure 3-1: Flash Memory Map 44
Figure 6-1: Platform Controller Hub (PCH) PIRQ to IRQ Router 66
Figure 6-2: Platform Controller Hub (PCH) / PCI Interrupt Routing Control 67
Figure 6-3: PCI Expansion Slot Interrupt Routing 70
Figure 7-1: Platform Controller Hub (PCH) DMI Link and Root Ports 85
Figure 8-1: Platform Controller Hub (PCH) Downstream Root Port Configuration 1 ... 97
Figure 8-2: L1 Total Exit Latency Calculation Example 103
Figure 8-3: Generation of SERR# to Platform 104
Figure 8-4: Platform Controller Hub (PCH) PCI Express* PME to SCI Path 106
Figure 8-5: Native Hot Plug Software Model 110
Figure 8-6: Platform Controller Hub - PCI Express* Hot Plug to SCI Path 111
Figure 8-7: ExpressCard* Slot Conceptual Diagram 115
Figure 8-8: BIOS Flow for PCIe* LTR Discovery and Enumeration 129
Figure 9-1: Intel® HD Audio Codec Node Structure and Addressing 131
Figure 18-1: BIOS POST Flow 239
Figure 22-1: Intel® Serial I/O Initialization Flow 262
Figure 23-1: ACPI Configuration Table 274
Figure 24-1: PCIe* Storage, reporting as AHCI single port device 279
Figure 24-2: OS View of the Remapped PCIe* Storage 280
Figure 24-3: Fast Boot/Sx Resume Handling 286

Tables

Table 2-1: Addressing Registers via SBREG_BAR 32
Table 2-2: Memory-Mapped Registers Affected by Posted-Write..... 36
Table 3-1: Boot BIOS Strap Definition 43
Table 3-2: Hardware Sequencer Instruction Set..... 47
Table 5-1: I/O blocks in the Intel PCH 53
Table 5-2: BIOS BARs 56
Table 5-3: Hidden Devices' BARs 56
Table 5-4: SPI BAR 56
Table 6-1: Platform Controller Hub (PCH) PIRQ Routing Table 64
Table 6-2: Platform Controller Hub / PCI Express* Root Port Interrupt Mapping for Downstream Devices 68
Table 6-3: Intel PCH DMI Assert_INTx Interrupt Mapping Scheme 70
Table 6-4: Typical NMI Sources 76
Table 6-5: Interrupt Message Address Format 84
Table 6-6: Interrupt Message Data Format 84
Table 7-1: Recommended TC/VC Mapping 86
Table 8-1. Device Grant SKL PCH-LP and PCH-H PCIe IOSF Channels 121
Table 8-2. Target Grant SKL PCH-LP and PCH-H PCIe IOSF Channels 121
Table 8-3. Remote Transmitter Preset Override Mode Settings 123
Table 8-4. Remote Transmitter Coefficient Override Mode Settings 123
Table 16-1. GPIO reset types 197
Table 17-1: Thermal Management Sensor Enabling.....209
Table 18-1: ICC MEI Messages Availability 216
Table 18-2: MKHI MEI Messages Availability 236
Table 18-3: Definition of MKHI MEI Header 236
Table 18-4: Definitions for ICC Command Request Message.....236
Table 18-5: Definitions for ICC Command Request Data 237
Table 18-6: Definitions for ICC Command Response Message.....237
Table 18-7: Definition of ICC Command Response Data.....237
Table 19-1: Intel® ME Enabled System Behavior After G3 Exit.....244
Table 19-2: Supported Deep Sx Power Policy Configurations.....249
Table 22-1 Intel® Serial IO (SKL/KBL PCH-LP) 261
Table 29-1: MKHI MEI Messages Availability 311
Table 29-2: Definition of MKHI MEI Header 311
Table 29-3: Definitions for HSIO Command Request Message 311
Table 29-4: Definitions for HSIO Command Response Message 312
Table 29-5: Definition of HSIO Command Response Data 312

Revision History

Document Number	Revision Number	Description	Revision Date
CDI/IBL# 549921	0.5.0	Initial release	July 2014
CDI/IBL# 549921	0.5.1	No major updates	July 2014
CDI/IBL# 549921	0.5.2	• Updated Section 4.3 CPU reference • Added footnote for eSPI port 80 enabling section 2.5.5 • Updated Figure 7-1 • Added as footnote the note for SKL PCH-LP Ax in section 7.2.4. • Added as footnote the note for SKL PCH-LP Ax in section 24.3.2 • Replaced AHCIBar by ABAR • Updated note at section 13.4.2 for USB2.0 and USB3.0 offsets reference. • Updated section 14.1 for SKL PCH-H and PCH-LP references. • Updated section 24.3.2 ABAR memory allocation. • Updated section 24.3.2 for PCie link Width and Speed	August 2014
CDI/IBL# 549921	0.6	• General cleanup on naming convention • Added New Sections 18.2.3 ICC MKHI MEI Messages and 29.3. HSIO MKHI MEI Messages • Added high level SATA init flow section 14.1, redefined ABAR memory space init section 14.1.3 • Updated PMC BIOS requirements section 19.5 • Updated DCI support section 13.6 • Added note to reflect UIP SMI defeature section 9 • Updated Static Frequency Switching Additional steps section 9.6.4 • Added missing setup for PWRMBASE + 624h register section 22.2 • Updated host debugger information in section 30.1.1 bullets 8 and 9	October 2014
CDI/IBL# 549921	0.6.2	• Updated MPHY/M-PHY by HSIO or ModPHY • New Section 19.14 PCI Express Output Clock Enable/Disable Control • Section 16.1.1 and 16.1.2 added GPIOs enable Wake capable information. • Section 13.6 added DCI auto mode support • New Section 2.2.4 for SBILOCK Support • Updated Section 22.6 Intel® Serial I/O PCI Mode Initialization • Updated section 8.15.1 bullet 3 sub-bullet d register programming "if not all devices support LTR"	November 2014

Document Number	Revision Number	Description	Revision Date
		• Removed 324h[7] bit from 8.15 section bullet 9 sub-bullet a • Updated content for IOSF Port configuration for SKL PCH-H and PCH-LP at Section 8.13 • Updated content at section 8.12 • Updated content for Intel Serial I/0 PCI Mode Initialization chapter 22.6 • Updated values on bullet 1 for Chapter 22.4 for supporting OS using the UART as debug port • General cleanup	
CDI/IBL# 549921	0.6.3	• Updated flow for Intel(R) TH (remote debugger support), chapter 30.1 and 30.1.1 • Added bullet number 6 in both sub-section under section 2.2.1 • Updated steps for xHCI Controller Setup at Section 13.4.1 • Updated Memory Bandwidth Throttling details at Section 17.5.1 • Added bullet 8 at Section 17.3.1 to accommodate LP changes related. • Overall update to Camera I/O Controller section from 26.1 to 26.4 • Removed section 26.2.1 for PCI mode no longer supported. • Added New Chapter 14.1.4.3 for RAID UEFI Driver (RST UEFI Drive) • General cleanup	December 2014

Document Number	Revision Number	Description	Revision Date
CDI/IBL# 549921	0.7.0	• Updated section 23.4.3 for 3rd Party IP and specify bit mask content and added an example for _DSM function 3 • Removed RCBA reference in chapter 9.2 • Updated Section 23.2 moved bullet 2 to 1 and update to accomodate Sub class code for DSP enabled. • Added New section 25.5 under Intel Sensor Hub to accomodate Intel Proprietary Wake support. • Added New section 25.6 under Intel Sensor Hub to accomodate New MEI message _PDT_UNLOCK_DATA. • Updated section 14.1.5, 14.1.6, 14.3 for SKL PCH-H and PCH-LP: POR values for SATA SIR registers programming, SATA Controller disabling and PCH SAT controller value for cold boot and S3/S4 path • Updated section 17.3.1 thermal condition for LP and H SKUs • Added note to highlight ME FW GPIO ownership vs host ownership to section 16.3 • Value updated for P2SB Endpoint Mask set at section 2.2.3. • Removed bullet 9 from section 30.1 for ITH Bypass step • New Section 8.14.1 added for PCIe* Gen 3 Static Equalization Mechanism • Removed bullet 6 for both section PCH-LP and PCH-H at section 14.5.1 • Updated Port 80h content and added new sections 2.5.5.1-2 • Standard naming convention cleanup for section 5.14.4 and 5.14.5	February 2015
CDI/IBL# 549921	0.7.2	• Updated 8.2.7 Section, added new PCIe cap list init flow • Updated Figure 23-1; added comment at end of section 23.4.1 for DMICx2/DMICx4 policies; added DspFeatureMask policy information • Updated programming steppings and added missing steps at Chapter 8 • Updated section 13.4.1 overall setup steps • Added Section 16.3.1 for GPIO Pad in Native Mode and made Pad Reset Configuration Section 16.3.2 • Updated value in step 2 under 28.4 section to match Silicon Code • Added Section 8.14.2 and 8.14.2.1 for PCIe Gen3 SWEQ flow and Device Reset respectively • Added New Section 16.3.3 for Pad Mode Enabling for PWM mode • Updated Section 12.1 xDCI Enabling and Disabling • Added New Chapter13.4.2 for xHCI SSIC Interface Configuration • Updated Table 16.7 to match PadRstCfg definition with SKL PCH EDS • Added GPI to GPE Mapping example in Chapter 16.1.1	April, 2015

Document Number	Revision Number	Description	Revision Date
CDI/IBL# 549921	0.8.1	• Updated values for xHCI MBAR + 816Ch (bullet 16) and xHCI MBAR + 80B0h(bullet 21) at section 13.4.1 • Added one new bullet in BIOS Region steps and one note on BIOS guard for Section 3.7 • Removed Ax Stepping support on Silicon Code, removed all related notes: whole section 5.14.6; footnote 8 for bullet 2, bullet 9 for bullet 33 at section 24.3.2 • Added Note related to NMI2SMI_STS at section 6.5.2 • Updated section 10.6, renamed GP27_Pin to LAN_WAKE terminology • Added footnote to port 60/64 SMI at section 13.2.1 • Added Second paragraph for content improvement at section 16.1.2 • Added content to xHCI row at Per Interface Throttling Actions (bullet 3) Table for section 17.5 • Added Note for BIOS Debug Trace Usage Recommendation • Updated Slow Ring oscilation clock gate at chapter 9.6.3	May, 2015

Document Number	Revision Number	Description	Revision Date
CDI/IBL# 549921	0.9.0	• Section 5.9.1.1 – Updated memory recommended static allocation. • Section 7.2.4 – Added additional PCH DMI and OP-DMI programming step 7.d. • Section 8.2.1 – Added note for spa~spc map to PCIe Controller 1~3. • Section 8.2.1.1 – Added note for SPA maps to PCIe Controller 1. • Section 8.2.1.1 – Updated power management steps description. • Section 8.2.1.1 – Added note for SPB maps to PCIe Controller 2. • Section 8.3 – Added L0s must be disabled for Gen3 link. • Section 8.3 – Updated L1 Sub-state extended capability register setting to align with Skylake Silicon Initialization Code. • Section 8.12 – Updated register programming for PCIe SSD supports non-common clock mode with SSC enabled. • Section 8.14 – Added note PCIe Gen3 ASPM L0s is not supported. • Section 8.15 – Updated step 13.a and 13.b registers setting. Updated clock gating programing in root port. Updated Common Clock Exit Latency programming. • Section 8.15.1 – Updated power optimization configuration. • Section 10.7 – Added Intel Gbe power gating setting. • Section 12.1 – Updated xDCI PM flow. • Section 13.2.1 – Removed port 60/64 SMI note to section 13.4.6. • Section 13.4.1 – Updated step3,10,22 register setting. • Section 13.4.6 – Added note for port 60/64 SMI moved from section 13.2.1. • Section 16.3.2 – Added register value for GPIO pad reset configuration. • Section 19.5 – Updated power management setting and added lock down PMC step.. • Section 19.10 – Updated enable clock gating programming step must refer to Skylake Silicon Initialization Code. • Section 19.13 – Updated Power Optimizer programming step must refer to Skylake Silicon Initialization Code. • Section 23.2 – Updated setting for Intel® SST enabled system that is required to support also Inbox HD Audio driver.	May, 2015

Document Number	Revision Number	Description	Revision Date
CDI/IBL# 549921	1.0.0	• Section 3.7 – Update note for BIOS Guard. • Section 8.15 - Enable squelch for GEN3 capable PCIe ports by set PCI offset 324[7] = 1b • Section 13.4.1 – Update step 17/29/30/31 settings. • Section 13.5 – Add XHCI D3 enabling chapter • Section 14.1.6 – Add condition - If no lanes assigned to SATA when in AHCI mode, then perform the disabling steps to function disable the SATA Controller • Section 14.3 – BIOS has to program the "Extended General Configuration Register" for each RST PCIe Storage Cycle Router. • Section 19.10 – Update registers setting for LPM ODD designed. • Section 19.15 – Add chapter LPC Power Saving Consideration. • Section 22.6.1 – Add bullet 6 - Resume from S3, clear Memory Space Enable. • Section 22.6.1.1 – Clarify Intel Serial driver support ACPI mode only for I2C4 and I2C5 controllers. • Section 28.4 – Add step3 - PCR[SCS] + 600h [BIT3] = 1b for PCH-LP stepping C1 onwards	June, 2015

Document Number	Revision Number	Description	Revision Date
CDI/IBL# 549921	1.1.1	• Section 7.2.2 – Update DMI link ASPM L0s must be disabled on Skylake-H/S platform. • Section 8.2.1.1 – Update step 2.b.ii. • Section 8.3 – Update DMI ASPM L0s must be disabled. • Section 8.15 – Add first PCIe RP in each controller setting • Section 9.6.6 – Add section for Intel HDA power management and clock gating requirement. • Section 11.13 – Add section 11.13 for additional programming requirement. • Section 12.1 – Add step4: xDCIBAR + C110h [1]=1 and remove step10. • Section 13.4.1 – Update xHCI Controller Setup step3 and step9. • Section 13.4.6 – BIOS must clear the current SMI status by write "1" to LPC PCI register offset 94h[11:8] before doing the 2nd 60/64 access. • Section 13.4.7.1 – Add this section for Additional Recommendation of Host Controller Reset • Section 14.1 – BIOS also need to perform SATA function disable steps if no lanes assigned to SATA controller when in AHCI mode. • Section 19.5 – Update Addition Power Management Programming. • Section 19.8.3 – Update Alternate Access Mode description. • Section 20.4.1 – Add section 20.4.1 for Additional SMI handling of NEW CENTURY of Skylake • Section 23.4.1 – Update feature mask table. • Section 24.3.2 – Update bullet 16 and 17. • Section 25.6.1 – Update PDT Unlock Data Message table. • Section 28.2.2 – Remove SDIO enable step since it is de-feature on Skylake platform. • Section 28.2.3 – Add SD Card specific programming • Section 29.2 – Remove ModPHY Lane Sus Well Power Gating since it's not POR.	June, 2015
CDI/IBL# 549921	1.2.1	• Section 2.3 – Add PCR target Port ID table. • Section 9.1.4 – Update step2 only apply for HDA link configured as 6MHz. • Section 9.6.3 – Add step 5,6,7,8. • Section 10.7.2 – Add the section for Clock and Lock Control Configuration. • Section 13.4.1 – Update USB2 setting for USB2 signal quality. • Section 19.16 – Add theis section for SPI/eSPI power management programming. • Section 22.4 – Add step3, dummy read is required after setting any of GPPRVRW7. • Section 26.2 – Add step2 to save power for CIO2 in disabling flow. • Section 26.4 – Update step10 for CIO2 clock gating enabled.	July, 2015

Document Number	Revision Number	Description	Revision Date
CDI/IBL# 549921	1.4.0	• Section 4 – Update platform reset control description. • Section 9.5 – Add disable Intel Display Audio description. • Section 9.6.4 – Update INPAY value for HDA 24MHz. • Section 10.7 – Add new section "Retrieve Lan Phy Revision" • Section 11.13 – Update SMBus PCI offset 64h setting. • • Section 19.5 – Add native power button disable BIOS setting. • Section 19.5 - Set PWRMBASE + offset 34Ch [21] = 1b for SPT-H DMI PHY PG Latency. Enable HSIO Power Gating for all data lanes on PchLp only. • Section 22.2 – Add setp3. • Section 22.8 – Update "Intel Serial I/O Power Management Programming". • Section 24.3.2 – Add step34 for allow legacy interrupts delivery. • Section 29.3 – Add section for HSIO Power Management programming for all data lanes and common lanes on PCH-LP only. • Section 28.5 – Add SD DLL programming • Section 28.6 – Update eMMC DLL programming. • Section 30.1 – Update step4.	August, 2015
CDI/IBL# 549921	1.6.0	• Section 8.2.1.1 – Update bullet 1.b. • Section 8.3 - PCIe L1.LOW must be disabled if CLKREQ is not assigned. • Section 8.15 – Step25 setting to align with Silicon Initialization Code. • Section 12.1 – Update xDCI disable flow. • Section 13.4 – Add XHCI compliance mode disable setting. • Section 13.4.1 – Remove xHCI MBAR + 8190h[14] programming. • Section 13.4.3 – Update USB3PDO and USB2PDO programming requirement. • Section 16.6.1 – Update GPP_A_12 description. • Section 19.16 – Add section for PSTH power management programming. • Section 22.2 – Update sideband mechanisim write access required, and add a Note for SerialIO does not support PME. • Section 23.3 – Update Programming interface value. • Section 24.3.2 – Remove step11 "Set "Port x Config Hide" to 1"	September, 2015
CDI/IBL# 549921	1.7.0	• Section 11.13 – Update SMBus register setting(SMBus PCI 64h = 0A0A0000h) that should be programmed in BIOS early phase. • Section 13.4.1 – Add item4 and item5. • Section 16.6.1 – Update GPP_A_12 handling description. • Section 16.8 – Update GPIO ACPI device ID for PCH-H.	October, 2015

Document Number	Revision Number	Description	Revision Date
CDI/IBL# 549921	1.8.0	• Section 9.6.2 – Update Intel HD Audio disable setting. • Section 9.6.3 – Update HDA IoBufferOwnership setting; BIOS has to program HDBAR + 08h[0] = 1 after assign I2S ownership. • Section 13.4.1 – Update XHCI programming step to align with Skylake Silicon Initization Code. • Section 13.4.2 – Add new section for Super Speed Port Enable register setting requirement. • Section 13.4.3 – Update xHCI SSIC interface configuration. • Section 14.1.3 – Update if remapping is enabled, BIOS has to disable Enclosure Management capability. • Section 19.5 – Update PWRMBASE + offset E0h setting for eSPI designed system. • Section 30.1 – Update ITH enable setting in item11.	November, 2015
CDI/IBL# 549921	2.0.0	• Added Kaby Lake branding • Standardized nomenclature and formatting • Section 1.3.7 - Updated guidance on DCI programming • Section 16 – Updated GPIO programming • Section 31 – Added for Lewisburg support	November, 2015
CDI/IBL# 549921	2.1.0	• Section 8.3 – Update guidance on PCIe L1 PM programming. • Section 16 - Updated GPIO programming references. • Section 14.1.6 – Removed step #1 programming SATA	January, 2016
CDI/IBL# 549921	2.1.1	• Section 8.3 – Update guidance on PCIe L1 PM programming.	January, 2016
CDI/IBL# 549921	2.1.2	• Section 8.3 – Update guidance on PCIe L1 PM programming. • Section 24.3.2 – Update RST initialization programming	January, 2016
CDI/IBL# 549921	2.2.0	• Updated Document Title. • 4.3.1 – Corrected typo in CPU only reset port 92h bit number. • 5.14.4 – Updated PSF EOI Multicast Configuration. • 5.14.6 - Added new power management setting. • 9.6.3 - Update SKL Audio PLL settings. • 12 – Update USB Type C nomenclature. • 13.4.1 – Updated xHCI Controller Setup. • 14.1.3 – Change SATA implemented bit. • 16.4.3.4 – Added GPIO new configuration instructions. • 17.3.1 – Updated PCHHOT# support. • 23.4.3 – Updated 3rd Party processing module bitmask.	May, 2016

Document Number	Revision Number	Description	Revision Date
CDI/IBL# 549921	2.3.1	• Section 5.14.4 – Updated PSF EOI Muticast Configuration. • Section 6.1.3 – Updated Root Port Interrupt Routing • Section 13.4 – Updated compliance mode support. • Section 16.9 – Updated GPIO reset guidance • Section 17 – Standarized references to ThermalBAR • Section 17.5 – Updated Throttling Configuration for KBP. • Section 19.5 – Updated Power management programming • Section 23.4.3 – Updated 3rd party IP module support superceding 23.4.3.1 • Section 23.4.3.1 – removed • Section 24.5 Added for RST UEFI Variable support	October, 2016
CDI/IBL# 549921	2.3.2	• Section 5.14.4 – Updated PSF EOI Muticast Configuration. • Section 19.5 – Updated Power management programming • Section 5.15 - Enable low latency mode for legacy IO	December, 2016
CDI/IBL# 549921	2.3.3	• Section 13.7 - Update DCI Lock after DCI enable/disable • Section 13.7 - DciAutoDetect is deprecated and removed • Section 31 – Update Lewisburg PCH	February, 2017
CDI/IBL# 549921	2.3.4	• Section 1.2 – Update CDI#557130 as new HID Event filter guide • Section 5.1 – Correct P2SB PCI offset 04h[1] MSE bit • Section 5.15 – Update Legacy IO latency mode • Section 8.2.1.1 – Update PCIe* Function Disable Handling • Section 8.8.2.2.1 – Update minor typo _STA()	November, 2017

§ §

1 Introduction

This document is intended to supplement the information provided in the Skylake-H Platform Controller Hub (PCH) External Design Specification (EDS), the Skylake-LP PCH EDS, the Kaby Lake-H PCH EDS, and the Kaby Lake-LP PCH EDS. It provides details needed to develop a System BIOS that supports the Skylake PCH H/S, the Skylake PCH-LP, the Kaby Lake PCH H/S, and the Kaby Lake PCH-LP. The current applicable EDS or data sheet should be used along with this document to get the correct register locations and current register and bit settings of the PCH registers.

This document describes the implementation of a full-featured System BIOS. The word “should” is used to describe recommendations categorized to take advantage of the capabilities of platforms that incorporate the PCH. The word “must” is used to describe functions that are required to ensure correct and reliable operation of platforms that incorporate the PCH.

1.1 Terminology

Term	Description
AHCI	Advanced Host Controller Interface, an industry specification of the interface between memory and SATA devices.
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.
ASF	Alert Standard Format, This is the next generation of “Alert on LAN*” implementation
Chipset Configuration Space	In PCH prior to SKL PCH context, the memory-mapped configuration register space whose base address is specified by RCBA register at D31:F0:F0h.
Core Power Well	Main system power, turns off in S3 – S5
CSI	Camera Serial Interface
DCI	Direct Connect Interface. Used for Closed Chassis Debug (DCI 2.0 introduces new set of features for Skylake and Kaby Lake PCH)
DMI	Proprietary data path between Processor and PCH
DMI Port	Also known as “North port”, the internal RCRB port at the PCH end of the DMI link.
DSP	Digital Signal Processor
eMMC	Embedded Multimedia Card
EDS	External Design Specification. For this platform, it refers to “"RS – Skylake/Kaby Lake Platform Controller Hub External Design Specification"
eSPI	Enhanced Serial Peripheral Interface
FPSBA	Flash PCH Base Address. Refer to PCH EDS for more details.
HC	Host Controller

Term	Description
LPC Bus	"Low Pin Count" connection used to connect to devices such as FHW, the super I/O device, and Trusted Platform Module (TPM).
I^2C	Inter-Integrated Circuit
I^2S*	Integrated Interchip Sound*
Intel® ME	Intel® Management Engine
Intel® RST	Intel® Rapid Storage Technology
Intel® Serial I/O	Refers to the following Interfaces: I2C, UART and GSPI
IOSF	Intel On-chip System Fabric
ITSS	Refers to Interrupt Configuration registers.
OPI	On-Package Interface
OTG	On The Go (USB)
P2SB	Primary-to-Sideband Bridge
PCHSTRP	PCH Soft Strap. Refer to related platform PCH SPI Programming Guide for more details.
Platform Reset	PCH asserts PLTRST# to reset devices that reside on the Primary PCI bus. The Intel PCH asserts PLTRST# during power-up and when a hard reset sequence is initiated through the CF9h register. 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 (PCIe* RP PCI offset 3Eh[6]).
PCR	Private Configuration Register
PEC	Packet Error Checking, This is a SMBus 2.0 feature
PG	Power Gate
PSF	Primary Scalable Fabric
PTI	Parallel Trace Interface
RCRB	Root Complex Register Block, as defined in PCI Express* Specification v1.0a. In PCH prior to Skylake context, it refers to a part of the Chipset Configuration space defined above.
RCBA	In PCH prior to Skylake context, Root Complex Base Address register at D31:F0:F0h. It specifies the physical address of Chipset Configuration space. Also used in or "RCBA + xxxxh" (where xxxxh is the offset) to indicate register location in the Chipset Configuration space.
Resume Power Well	Trickle from power supply, only turns off when power removed from wall
Resume Reset	Signal that resets the parts of the PCH in the resume power well, generated when the trickle supply turns on.
RO	Read-Only
RWO	Read-Write-Once

Term	Description
RTC Power Well	Powered by coin cell battery, only turns off when battery is drained. Powers RTC and some resume events
RTCRST#	Signal that resets the RTC well (but does not clear the RTC RAM memory contents)
SA	System Agent
SATA	Serial ATA, an industry specification of the interface for storage controllers and devices.
SB	Sideband (see below)
SBI	Sideband Interface.
South Port	The PCI Express* downstream root port(s) on PCH.
SPI	Serial Peripheral Interface
SSIC	SuperSpeed InterChip
TCO	"Total Cost of Ownership", features that help manage the platform
UART	Universal Asynchronous Receiver/Transmitter
VLM	Virtual Legacy Wire Message

1.2 Related Documents

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

- Skylake Platform Controller Hub (PCH) External Design Specification (EDS)
- Kaby Lake PCH EDS
- SKL and PCH-H volume 1 and 2, CDI/IBL documents#: 546717 and 546955
- SKL and KBL PCH-LP volume 1 and 2, CDI/IBL documents#: 545659 and 545845
- Skylake and Kaby Lake PCH SPI Programming Guide
- Intel® Management Engine 11 BIOS Specification for Skylake Platform (CDI/IBL document# 549522)
- BIOS Enabling Guide for Win10 technical white paper(CDI/IBL documents# 557130)
- Serial ATA Advanced Host Controller Interface (AHCI) Specification Rev 1.3.1 http://www.intel.com/content/www/us/en/io/serial-ata/serial-ata-ahci-spec-rev1-3-1.html
- Pin Configuration Guidelines for High Definition Audio Devices http://www.microsoft.com/whdc/device/audio/PinConfig.mspx
- Microsoft HD Audio Pin Configuration Implementation Test Procedure Guidelines http://www.microsoft.com/whdc/device/audio/PinConfig_test.mspx
- Intel® 82093AA I/O Advanced Programmable Interrupt Controller (I/O APIC) Datasheet http://www.intel.com/design/chipsets/datashts/290566.htm
- Multiprocessor Specification, Version 1.4 http://www.intel.com/design/archives/processors/pro/docs/242016.htm

- Microsoft Corporation, Simple Boot Flag Specification, Version 2.0
  http://www.microsoft.com/whdc/resources/respec/specs/simp_boot.mspx
- Plug and Play BIOS specification, Version 1.0a
  http://www.microsoft.com/whdc/system/pnppwr/pnp/default.mspx
- Advanced Configuration and Power Interface (ACPI), Revision 4.0a
  http://www.acpi.info/spec.htm
- PCI BIOS 2.1 and PCI Firmware 3.0
  http://www.pcisig.com/specifications/conventional/pci_firmware/
- Conventional PCI 3.0
  http://www.pcisig.com/specifications/conventional/pci_30/
- PCI to PCI Bridge Architecture 1.2
  http://www.pcisig.com/specifications/conventional/pci_to_pci_bridge_architecture
- PCI Express* Base Specification 1.1
  http://www.pcisig.com/specifications/pciexpress/base/
- Alert Standard Format (ASF) Specification 2.0
  http://www.dmtf.org/standards/asf/
- System Management BIOS Reference Specification 2.5
  http://www.dmtf.org/standards/smbios/
- System Management Bus (SMBus) Specification 2.0
  http://www.smbus.org/specs/index.html
- Universal Serial Bus Mass Storage Class Control/Bulk/Interrupt (CBI) Transport Specification 1.1
  http://www.usb.org/developers/devclass_docs/usb_msc_cbi_1.1.pdf
- Requirements for Hardware Watchdog Timers Supported by Microsoft* Windows* design specification
  http://www.microsoft.com/whdc/system/sysinternals/hw-wdt.mspx
- Debug Port Specification, Version 1.0a
  http://www.microsoft.com/whdc/archive/debugspec.mspx
- IA-PC HPET (High Precision Event Timers) Specification
  http://www.intel.com/hardwaredesign/hpetspec_1.pdf
- PCIe* GbE Controllers Open Source Software Developer's Manual
  http://download.intel.com/design/network/manuals/316080.pdf
- Multifunction Device Support and Device Container Groupings in Windows* 7
  http://www.microsoft.com/whdc/Device/DeviceExperience/ContainerIDs.mspx
- Skylake Performance Tuning Guide #559338

§ §

2 Intel Platform Controller Hub (PCH) Overview

The Skylake-H PCH provides extensive I/O support. Functions and capabilities include:

- PCI Express* Base Specification, Revision 3.0 support for up to 20 lanes and 16 ports with transfers up to 8 GT/s
- ACPI Power Management Logic Support, Revision 4.0a
- Interrupt controller, and timer functions, without timer 1
- Integrated Serial ATA host controllers with independent DMA operation on up to 8 ports. SATA Express support on all ports.
- xHCI USB controller provides support for up to 12 USB ports, of which 10 can be configured as SuperSpeed USB 3.0 ports. Supports new USB debug port with DCI support.
- Flexible IO - A new architecture to allow some high speed IO signals to be configured as PCIe*, GbE, SATA or USB 3.0.
- Integrated 10/100/1000 Gigabit Ethernet MAC with System Defense
- System Management Bus (SMBus) Specification, Version 2.0 with additional support for I^2C devices
- Supports Intel® High Definition Audio (Intel® HD Audio)
- Supports Intel® Audio Digital Signal Processor (DSP)
- Supports Intel® Rapid Storage Technology (Intel® RST)
- Supports Intel® Active Management Technology (Intel® AMT)
- Supports Intel® Virtualization Technology for Directed I/O (Intel® VT-d)
- Supports Intel® Trusted Execution Technology (Intel® TXT)
- Integrated Clock Controller
- Low Pin Count (LPC) interface
- Firmware Hub (FWH) interface support
- One Serial Peripheral Interface (SPI) support for flash/TPM
- One Serial Peripheral Interface (SPI) support for devices like sensor
- eSPI
- JTAG Boundary Scan support
- Two integrated general purpose SPI controllers
- Two integrated I^2C controllers
- Two integrated UART controllers
- Integrated Sensor Hub
- Fan Speed Control

Note: Not all functions and capabilities may be available on all SKUs. Please refer to the latest PCH EDS or Data Sheet for details on SKU feature availability.

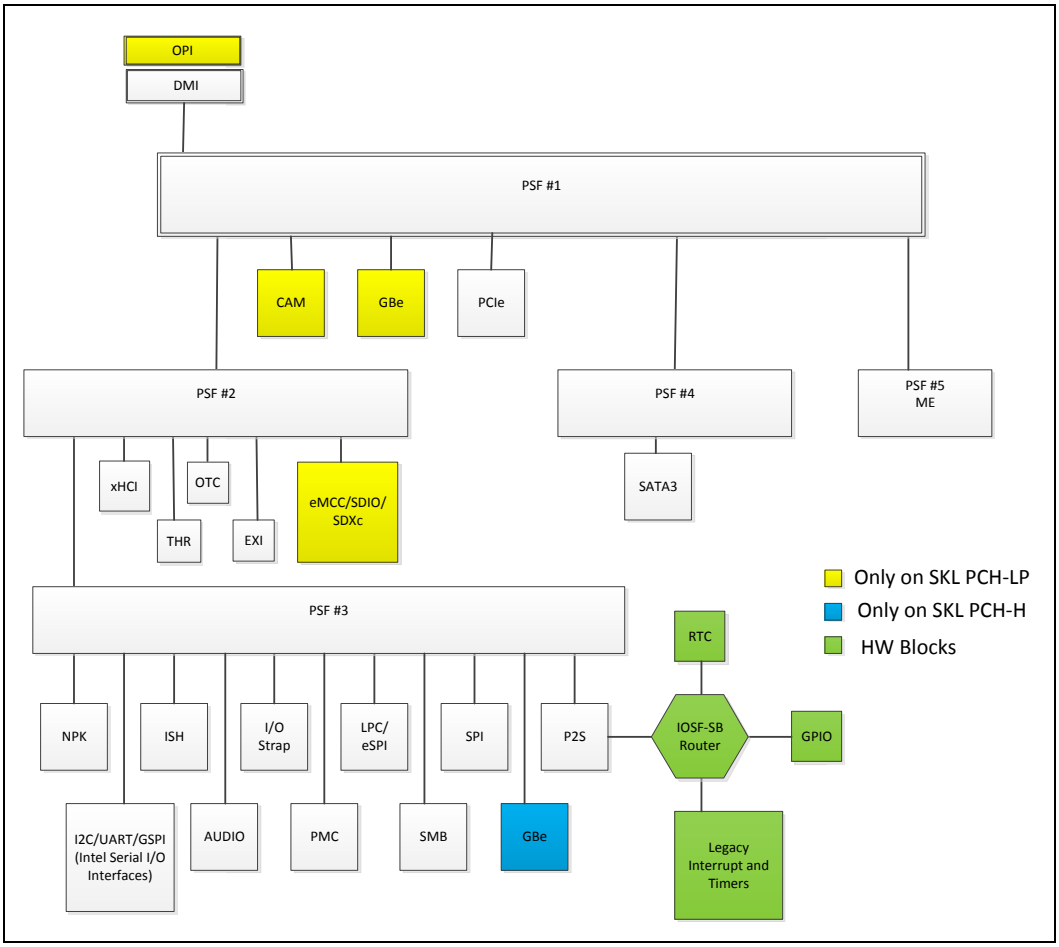
The Skylake-LP PCH provides extensive I/O support. Functions and capabilities include:

- PCI Express* Base Specification, Revision 3.0 support for up to 12 lanes and six ports with transfers up to 8 GT/s

- ACPI Power Management Logic Support, Revision 4.0a
- Interrupt controller, and timer functions, without timer 1
- Integrated Serial ATA host controllers with independent DMA operation on up to three ports. SATA Express support on all ports
- xHCI USB controller provides support for up to 10 USB ports, of which 6 can be configured as SuperSpeed USB 3.0 ports. Up to two could be mapped as SSIC ports. New USB debug port support with DCI support and USB Device controller and OTG support
- Flexible IO - Allows some high speed IO signals to be configured as PCIe*, CSI2, GBE, USB-SSIC, SATA or USB 3.0
- Integrated 10/100/1000 Gigabit Ethernet MAC with System Defense
- System Management Bus (SMBus) Specification, Version 2.0 with additional support for I2C* devices
- Supports Intel® High Definition Audio (Intel® HD Audio)
- Supports Intel® Audio Digital Signal Processor (DSP)
- Supports Intel® Rapid Storage Technology (Intel® RST)
- Supports Intel® Active Management Technology (Intel® AMT)
- Supports Intel® Virtualization Technology for Directed I/O (Intel® VT-d)
- Supports Intel® Trusted Execution Technology (Intel® TXT)
- Integrated Clock Controller
- Low Pin Count (LPC) interface
- Firmware Hub (FWH) interface support
- One Serial Peripheral Interface (SPI) support for flash/TPM
- One Serial Peripheral Interface (SPI) support for devices like sensor
- eSPI
- JTAG Boundary Scan support
- Two integrated general purpose SPI controllers
- Six integrated I^2C controllers
- Three integrated UART controllers
- Camera Sensor Interface
- Integrated Sensor Hub
- eMMC/SDIO/SD

Not all functions and capabilities may be available on all SKUs. Please refer to the latest PCH EDS or Data Sheet for details on SKU feature availability.

Figure 2-1: Skylake / Kaby Lake PCH-LP and PCH-H Block Diagram



2.1 Platform Controller Hub (PCH) Internal PCI Devices and Functions

The PCH incorporates a variety of PCI devices and functions. In these devices, it could have registers that are accessed through the primary interface, such as PCI configuration space, IO/MMIO space. There are also registers put under the private configuration space that is accessible through sideband interface.

Please see EDS for more detail.

2.2 Primary to Sideband Bridge (P2SB)

One of the major features of this bridge is to provide a mechanism for transactions to cross between the Primary and Sideband networks. This allows host processor access to resources residing on the sideband network.

A 16-MB aperture is provided in 64b memory space that allows direct mapping to 16 bits of DWORD register space for each of the 255 sideband devices. This memory BAR

that defines the memory location for providing register accesses will be referred to as "SBREG_BAR".

Note: It is likely that BIOS would assign the BAR to be under 4 GB for 32-bit OS ACPI usage, beside some existing BIOS framework such as IO trap SMI handling expect address to be at most 4 bytes.

Table 2-1: Addressing Registers via SBREG_BAR

Addr[63:24]	Addr[23:16]	Addr[15:2]	Addr[1:0]
REG_BAR[63:24]	TargetPort[7:0]	TargetRegister[15:2]	

Table 2-1 shows how a system address ("Addr") for a MMIO transaction is interpreted for accessing chipset registers. The TargetPort field defines the 8b Port ID and the TargetRegister field defines the 16b register address that will be used when sending the transaction to sideband.

Note: All accesses to the SBREG_BAR region are DWORD aligned DWORDS. If the lower two bits of the address are not 0, make sure the access does not cross DWORD boundary. Writes larger than one DWORD will be completed normally but write data beyond the first DWORD will be discarded. Reads larger than one DWORD will result in an Unsupported Request (UR) response.

SBI access registers are also located in this bridge. Refer to section 2.2.1 for its programming steps. However, it is recommended to use the MMIO access method instead for private configuration register access which is faster and atomic. Refer to section 2.6 for exceptions.

2.2.1 SBI Programming

In order to read registers in private register space using SBI Index Data pair, BIOS needs to follow steps below for read:

1. Poll P2SB PCI offset D8h[0] = 0b
2. Write P2SB PCI offset D0h[31:0] with Address and Destination Port ID.
3. Write P2SB PCI offset DCh[31:0] with extended address, which is expected to be 0 for private configuration register in the PCH.
4. Write P2SB PCI offset D4h[31:0] with dummy data such as 0, because all D0-DFh register range must be touched in the PCH for a successful SBI transaction.
5. Set P2SB PCI offset D8h[15:8] = 00000110b
6. Must clear P2SB PCI offset D8[7] = 0b
7. Write P2SB PCI offset DAh[15:0] = F000h
8. Set P2SB PCI offset D8h[0] = 1b, Poll P2SB PCI offset D8h[0] = 0b
9. Read P2SB PCI offset D8h[2:1] = 00b for successful transaction
10. Read P2SB PCI offset D4h[31:0] for SBI data

In order to write to registers in private register space using IOSF SBI Index Data pair, BIOS needs to follow steps below:

1. Poll P2SB PCI offset D8h[0] = 0b
2. Write P2SB PCI offset D0h[31:0] with Address and Destination Port ID.

3. Write P2SB PCI offset DCh[31:0] with extended address, which is expected to be 0 for private configuration register in the PCH.
4. Write P2SB PCI offset D4h[31:0] with the intended data accordingly
5. Set P2SB PCI offset D8h[15:8] = 00000111b
6. Set P2SB PCI offset D8[7] = 0b for non-posted transaction (limited IP support posted transaction, refer to SKL PCH EDS for details)
7. Write P2SB PCI offset DAh[15:0] = F000h
8. Set P2SB PCI offset D8h[0] = 1b
9. Poll P2SB PCI offset D8h[0] = 0b
10. Read P2SB PCI offset D8h[2:1] = 00b for successful transaction

Note: Please refer to EDS for P2SB PCI offset D8h[15:8] definition when accessing register space other than private register space. It is possible that the extended address would need to be set to none zero value if the related address exceeds 16 bits.

2.2.2 MMIO Back to Back Write Restriction and Handling

BIOS must issue a read after each write to ensure that the write has completed before issuing the next write. For GPIO and ITSS that support sideband posted write, they can support back to back write after their corresponding bits under P2SB PCI Config 80h-9Fh are set.

Program the following at early PCH BIOS init

1. Set P2SB PCI offset 80h to 0
2. Set P2SB PCI offset 84h to 0
3. Set P2SB PCI offset 88h to 0
4. Set P2SB PCI offset 8Ch to 0
5. Set P2SB PCI offset 90h to 0
6. Set P2SB PCI offset 94h to 0000F000h
7. Set P2SB PCI offset 98h to 00000010h
8. Set P2SB PCI offset 9Ch to 0

2.2.3 Endpoint Mask

Endpoint masks provide a mechanism for disabling particular IOSF-SB endpoints from being allowed to be targeted by transactions from the P2SB. The BIOS should set P2SB PCI offset C4h [29, 28, 27, 26] to [1, 1, 1, 1].

Prior to any 3rd party code running, the BIOS has to set the “Endpoint Mask Lock”, P2SB PCI offset E2h bit[1] to 1.

2.2.4 SBILOCK

The SBILOCK, P2SB PCI offset E3h bit [7], will lock down access to the P2SB SBI registers (P2SB PCI offset D0h-DFh). The BIOS is to set this bit to lock down the SBI accesses prior to 3rd party code running. The implication is that there must be no runtime SBI access such as in SMM.

2.3 Memory-Mapped Chipset Configuration Space

Starting from Skylake PCH onwards, most of the previous memory-mapped Chipset Configuration space, RCRB, is relocated into multiple “chipset register” ranges at individual IP sideband endpoint, in such they are now accessible through the sideband mechanism or the MMIO access mechanism shown in 2.2. Some RCRB registers are also relocated to MMIO space under different PCI devices (SPI, PMC).

Throughout this document, “PCR[TargetPort] + xxxx” will be used to indicate register locations in this sideband chipset private configuration register space, where xxxx is the register offset from the base address specified by the SBREG_BAR register and the target port id. The terms used in “TargetPort” is sometime abstracted with the name used in the EDS. For the PSF PCR registers, sometime abstracted name is used for the xxxx offset, for example, PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[ISH] is referring to ISH’s "AGNT_T0_SHDW_PCIEN" register under PSF3; in both Skylake and Kaby Lake PCH EDS, this would be PSF_3_AGNT_T0_SHDW_PCIEN_RS0_D19_F0 = 81Ch

Skylake and Kaby Lake Private Configuration Register(PCR) Target PortID

Private Configuration Register(PCR)		
PCH Device/Function Type	**Name Definition in Silicon Initialization Code**	**Target Port ID Number**
DMI Configuration **(SPT-H ONLY)**	PID_DMI	EFh
OPI Configuration **(SPT-LP ONLY)**	PID_OPIPHY	ECh
HSIO Strap Configuration	PID_PSTH	89h
eSPI / SPI	PID_ESPISPI	EEh
Real Time Clock (RTC)	PID_RTC	C3h
Processor Interface, 8254 Timer, HPET, APIC	PID_ITSS	C4h
SMBus	PID_SMB	C6h
PSF1 - Function Disable	PID_PSF1	BAh
PSF2 - Function Disable	PID_PSF2	BBh
PSF3 - Function Disable	PID_PSF3	BCh
PSF4 - Function Disable	PID_PSF4	BDh
General Purpose I/O (GPIO) Community 0	PID_GPIOCOM0	AFh
General Purpose I/O (GPIO) Community 1	PID_GPIOCOM1	AEh
General Purpose I/O (GPIO) Community 2	PID_GPIOCOM2	ADh
General Purpose I/O (GPIO) Community 3	PID_GPIOCOM3	ACh
PCIe Controller #1	PID_SPA	E0h
PCIe Controller #2	PID_SPB	E1h

Private Configuration Register(PCR)		
PCH Device/Function Type	**Name Definition in Silicon Initialization Code**	**Target Port ID Number**
PCIe Controller #3	PID_SPC	E2h
PCIe Controller #4 **(SPT-H ONLY)**	PID_SPD	E3h
PCIe Controller #5 **(SPT-H ONLY)**	PID_SPE	E4h
Integrated Clock Controller	PID_ICC	DCh
HDA / DSP	PID_DSP	D7h
USB Dual Role / OTG	PID_OTG	E5h
USB2.0	PID_USB2	CAh
DCI	PID_DCI	B8h
eMMC/SDXC **(SPT-LP ONLY)**	PID_SCS	C0h
UART, I2C, GSPI Interface	PID_SERIALIO	CBh
ISH Controller	PID_ISHBR	BFh
CSI-2 DFx **(SPT-LP ONLY)**	PID_CAM_FLS	AAh
CSI-2 Interface **(SPT-LP ONLY)**	PID_CAM_CHC	A1h
FIA Configuration	PID_FIA	CFh
SSIC Controller **(SPT-LP ONLY)**	PID_MMP0	B0h
MODPHY0 (HSIO Lanes 1-6)	PID_MODPHY0	EAh
MODPHY1 **(SPT-H: HSIO Lanes 7-14)** **(SPT-LP: HSIO Lanes 7-10)**	PID_MODPHY1	E9h
MODPHY2 **(SPT-H: HSIO Lanes 15-18)** **(SPT-LP: HSIO Lanes 11-16)**	PID_MODPHY2	A9h
MODPHY3 **(SPT-H: HSIO Lanes 19-26)**	PID_MODPHY3	A8h
SATA	PID_SATA	D9h
XHCI	PID_XHCI	E6h
LPC	PID_LPC	C7h

Note: Please refer to Skylake or Kaby Lake PCH EDS for more PCR registers information

2.3.1 Memory-Mapped Registers Affected by Posted-Write

The memory-mapped configuration register space in the PCH can present conditions where posted writes may not be visible at the destination register prior to subsequent traffic dependent on the posted write taking effect. An example of this is a memory-mapped configuration register that enables decode of specific memory or I/O ranges

by PCH. Such memory or I/O cycles may not be claimed and sent to the correct destination if the prior write to the configuration register has not arrived at its destination register. Reading registers in this category provides a coherency-enforcing event that ensures the write to a memory-mapped register is complete. Note that I/O mapped registers naturally have the coherency check in place because the write cycle will not finish until the register update is complete.

It is recommended that System BIOS read back these register bits after writing to them to ensure the above-mentioned coherency.

The table below lists PCH memory-mapped configuration registers that are exposed to such conditions.

Table 2-2: Memory-Mapped Registers Affected by Posted-Write

	Affected Register	Notes
Serial IO devices MBAR1	ALL	Ensure PMCSR register is written

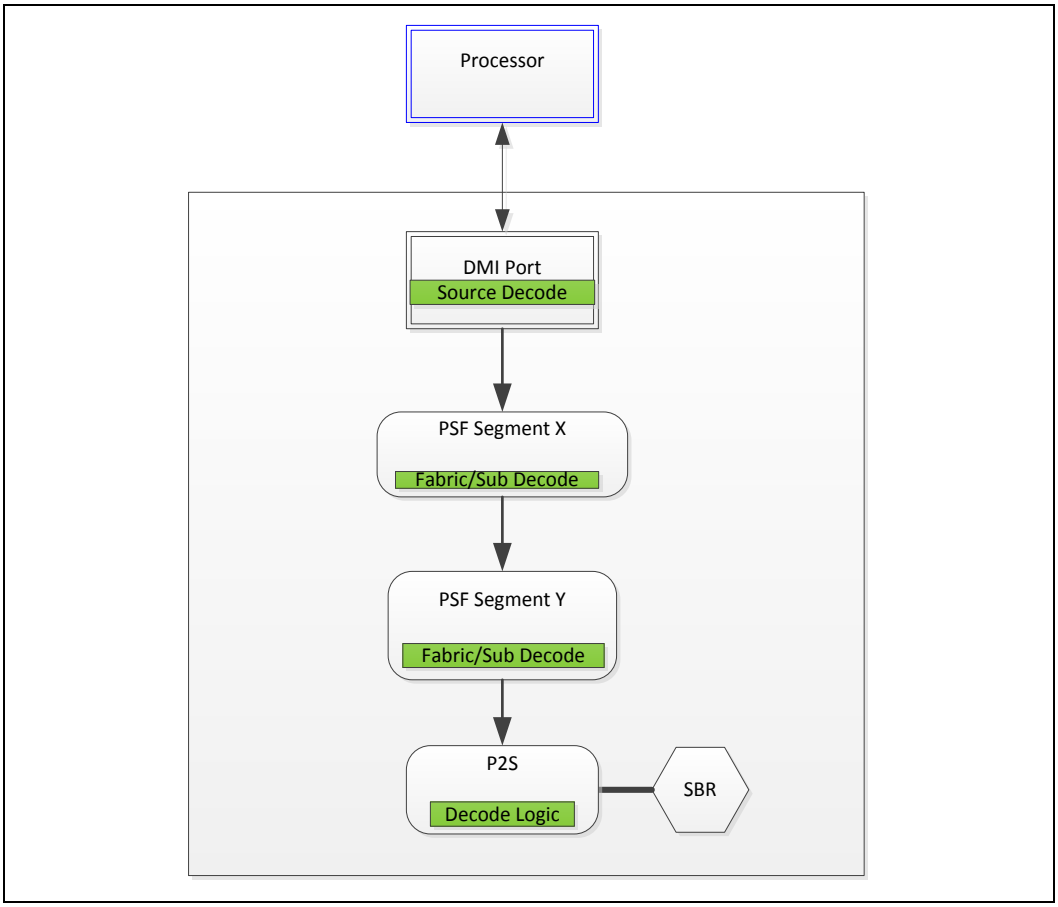
2.4 Platform Controller Hub (PCH) Configuration Register Changes

It is important to note that there is a revamp of access mechanism to these configuration registers from the earlier Chipsets before Skylake PCH. Care must be taken so that the necessary System BIOS code modifications are made to match such changes in the register/bit locations. While effort is made to keep some offset the same, some inevitably need to change. The EDS contains the complete list of register bit definitions.

2.5 Platform Controller Hub (PCH) Cycle Decoding

Figure 2-2 below illustrates the cycle decode of the PCH. All PCH internal functions are connected to the internal bus. Cycles received from the DMI interface will be decoded and claimed by the appropriate internal targets or, if unclaimed, forwarded to the PSF for fabric decode, to the P2SB for subtractive decoding.

Figure 2-2: Platform Controller Hub (PCH) Cycle Decoding



2.5.1 DMI Cycle Decoding

DMI is responsible to perform Source Decode on selective proprietary memory and I/O ranges claimed by LPC, eSPI, SPI Flash, PMC, Thermal Throttling block, except the Fixed I/O ranges. Downstream messages from processor with “Terminate at Receiver” routing ID will also be decoded and the message will be forwarded to PSF with the correct destination ID. Accesses to the different LaGrande Technology address spaces will also be handled through DMI source decode. Cycles that are not positively decoded by DMI source decode or fabric decode will be subtractive forwarded to the P2SB.

All the registers relevant to DMI source decode will be duplicated locally within DMI Private Configuration space. BIOS is responsible to keep the value of the registers up to date with the corresponding registers on the target side.

Trapping Port 60h/64h I/O cycles (USB Legacy Keyboard/Mouse Control) using the trap defined in LPC block is only possible if IOE.KE (LPC/eSPI offset 82h [10]) range is enabled.

All source decode programming except the IO Trap, has to be done before they are locked out by DMIC.SRL (PCR[DMI] + 2234h [31] = 1).

2.5.1.1 TBARB Decoding

It is recommended not to use TBARB, refer to section 17.2

If it is used however, setup TTTBARB and TTTBARBH registers, PCR[DMI] + 2724h and PCR[DMI] + 2728h to the same value program in Thermal PCI offset 40h and 44h.

2.5.1.2 LPC Generic I/O Range Decoding

Setup “LPC Generic I/O Range #”, PCR [DMI] + 2730h~273Fh to the same value program inLPC/eSPI PCI offset 84h~93h.

2.5.1.3 LPC Memory Range Decoding

Setup “LPC Memory Range”, PCR[DMI] + 2740h to the same value program in LPC/eSPI PCI offset 98h.

2.5.1.4 LPC BIOS Decoding

Setup “LPC BIOS Decode Enable”, PCR[DMI] + 2744h to the same value program in LPC/eSPI PCI offset D8h or SPI offset D8h depending on GCS.BBS pointing to LPC/eSPI or SPI.

2.5.1.5 LPC IO Decode Ranges Decoding

Setup “LPC I/O Decode Ranges”, PCR[DMI] + 2770h [15:0] to the same value program in LPC/eSPI PCI offset 80h.

2.5.1.6 LPC IO Enables Decoding

Setup “LPC I/O Enables”, PCR[DMI] + 2774h [15:0] to the same value program in LPC/eSPI PCI offset 82h.

Note: Bit[15:10] of the source decode register is Read-Only. The IO range indicated by the Enables field in LPC/eSPI PCI offset 82h[13:10] is always forwarded by DMI to subtractive agent for handling.

2.5.1.7 uCode Patch Region Decoding

Setup “uCode Patch Region Enable”, PCR [DMI] + 2748h[0] to ‘0b’.

2.5.1.8 Boot BIOS Strap

Refer to section 3.2.

2.5.1.9 I/O Trapping

Setup DMI “I/O Trap Register #”, PCR[DMI] + 2750h~276Fh to the same value program in “I/O Trap Register #”, PCR[PSTH] + 1E80h~1E9Fh.

2.5.1.10 TCOBASE Decoding

Setup “TCO Base Address”, PCR [DMI] + 2778h[15:5, 1] to [SMBUS PCI offset 50h[15:5], 1].

2.5.1.11 ABASE Bar Decoding

Program “ACPI Base Address”, PCR[DMI] + 27B4h[23:18, 15:2, 0] to [3Fh, Value, 1] where ‘Value’ is from PMC PCI offset 40h bit[15:2].

For Skylake and Kaby Lake PCH-LP:

Program “ACPI Base Destination ID”, PCR[DMI] + 27B8h[31:0] to [23A0h].

For Skylake and Kaby Lake PCH-H:

Program “ACPI Base Destination ID”, PCR[DMI] + 27B8h[31:0] to [23A8h].

2.5.1.12 PMCMBASE Bar Decoding

Program “PM Base Address Memory Range Base”, PCR[DMI] + 27ACh[15:0] to the same value programmed in PMC PCI offset 48h bit[31:16]; this has an implication of making sure the PWRMBASE to be 64-KB aligned.

Program “PM Base Address Memory Range Limit”, PCR[DMI] + 27ACh[31:16] to the value programmed in PMC PCI offset 48h bit[31:16], this has an implication of making sure the memory allocated to PWRMBASE to be 64KB in size.

For Skylake and Kaby Lake PCH-LP:

Program “PM Base Control”, PCR[DMI] + 27B0h[31, 30:0] to [1, 23A0h].

For Skylake and Kaby Lake PCH-H:

Program “PM Base Control”, PCR[DMI] + 27B0h[31, 30:0] to [1, 23A8h].

2.5.2 PSF Cycle Decoding

PSF is responsible fabric decoding, which in Skylake and Kaby Lake PCH, means subtractive decoding. In this mode, each PSF segment positively decodes downstream transactions for the agents that are in that segment, and subtractively sends a transaction to its subordinate PSF in case of no hit.

Whenever a function is to be disabled, it needs to be disabled at the IP/function level first, then at the PSF level register: Set PSF agent’s “PCI Configuration space enable bits” register’s “Function Disable” bit using byte access if the PCI configuration, IO space, memory space and bus master are to be disabled. If only the PCI configuration space is to be hidden, then set the “PCI Configuration Disable” register’s “Configuration Disable” bit.

It is possible for an IP/function default to disabled because of the soft strap. It is the BIOS role to program the relevant “Function Disable” bit before the IP (if it is part of the multi-function device) is accessed by BIOS (mostly due to PCI enumeration). Some exception to this as certain IP has the function disable register within its configuration space.

Refer to section 5.3 for full list of function disable BIOS actions.

PSF can also be used for mapping the BIOS BARs that are not source decoded by the DMI.

2.5.2.1 Intel® TH BIOS BAR Decoding

The third 64-bit memory BAR of Intel TH is a non-standard ACPI BAR that is residing at Intel TH PCI offset 70h. This ACPI BAR is shadowed by PSF3 under D20:F4, for which automatic configuration cycle snoop is not applicable. Upon power-up, BIOS will send configuration cycles to Intel TH for the ACPI BAR configuration. After that, BIOS must then manually write to the PCR[PSF_3] + "AGNT_T0_SHDW_BAR0"[D20:F4] and PCR[PSF_3] + "AGNT_T0_SHDW_BAR1"[D20_F4] to program the same BAR values, and also manually write 1 to PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[D20_F4] [1] to activate the shadow. D20:F4 is a phantom BDF only to enable this manual ACPI shadow, and therefore, BIOS must set configuration-disable for the function to hide it after the manual configuration is done.

2.5.3 LPC Cycle Decoding

Depending on the platform design, the following configuration registers must be programmed properly by System BIOS for PCH to positively decode LPC cycles targeted to the Super IO or firmware hub (FWH) on the LPC/eSPI (mutual exclusive, sharing using same BDF):

LPC IO Decode Range/Enable Register	LPC/eSPI PCI offset 80h-83h
FWH and BIOS Decode/Control registers (LPC)	LPC PCI offset D0h-DCh
BIOS Decode/Control registers (eSPI)	eSPI PCI offset D8h-DCh

For example, a platform has a Super I/O component on the LPC interface. To allow I/O cycles targeted to the keyboard controller, floppy controller, parallel port and serial ports in the Super I/O to be decoded by the PCH LPC/eSPI interface, the following bits of the LPC_EN register at LPC/eSPI PCI offset 82h must be set to 1:

Bit12 *Enable Decoding ranges to LPC (SE)*

Bit10 *Keyboard Enable locations to LPC (KE)*

Bit3 *Floppy Drive Enable range to LPC (FDE)*

Bit2 *Parallel Port Enable range to LPC (PPE)*

Bit1 *Com Port B Enable range to LPC (CBE)*

Bit0 *Com Port A Enable range to LPC (CAE)*

2.5.4 LPC Memory Cycle Decoding

Current LPC bridge will only positive decode cycles fall within FWH and LT memory range and will run these cycles as FWH and LT memory cycles respectively. No standard Memory Cycle will be forwarded to LPC/eSPI interface.

The PCH provides a generic memory range for decoding memory cycles and forwarding them as standard LPC Memory cycles on the LPC/eSPI. The generic memory decode range is 64-KB is size, locatable anywhere within the 4-GB space.

BIOS will configure this range during POST to provide the memory resources needed and will advertise the range to OS as Reserved Memory to avoid any resource conflict.

Register Name	Location	Description
LPC I/F Generic Memory Range	LPC/eSPI PCI offset 98h[31:0]	LGMR – LPC I/F Generic Memory Range

2.5.5 I/O Port 80h Cycle Decoding

2.5.5.1 PCIe Port 80h Cycle

I/O port 80h is typically used by decoder/LED hardware for debug purposes. By default PCH will forward IO port 80h cycles to LPC/eSPI[1] bus. The Reserved Page Route (RPR) bit of General Control and Status register, located at PCR[DMI] + 274Ch[11] , allows software to re-direct IO port 80h cycles to PCIe* bus so that a target (for example, a debug card) on PCIe* bus can receive and claim these cycles. The "RPR Destination ID", PCR[DMI] + 274Ch[31:16] need to be set accordingly to point to the root port that decode this range. Reading from Port 80h may not return valid values if the POST-card itself do not shadow the writes. Unlike LPC, PCIe* does not shadow the Port 80 writes.

2.5.5.2 LPC/eSPI Port 80h Word and Dword Cycles Workaround

The current PCH by default will only forward port 80h byte cycle to LPC/eSPI. If more than one byte post code is desired, the LPC generic I/O range would be used as workaround.

Note: It is assumed that if the eSPI is being used, the post code is routed to the slave 0. Applying the workaround on slave 1 would cause it to lose its one and only LPC generic I/O range.

For Skylake and Kaby Lake PCH-LP, the RP# destination IDs are given as below:

RP	Destination ID	RP	Destination ID
1	2188h	7	219Ah
2	2189h	8	219Bh
3	218Ah	9	21A8h
4	218Bh	10	21A9h
5	2198h	11	21AAh
6	2199h	12	21ABh

For Skylake Kaby Lake PCH-H, the RP# destination IDs are given as below:

[1] If eSPI instead of LPC is being used, the eSPI EC is expected to have support for the port 80h decoding. Please contact the related EC vendor for further enabling needs.

RP	Destination ID	RP	Destination ID
1	2180h	11	219Ah
2	2181h	12	219Bh
3	2182h	13	21A8h
4	2183h	14	21A9h
5	2188h	15	21AAh
6	2189h	16	21ABh
7	218Ah	17	21B8h
8	218Bh	18	21B9h
9	2198h	19	21Bah
10	2199h	20	21BBh

Note: Setting the RPR bit alone is not sufficient. Further programming is required on the PCIe* RP, refer to section 8.10.

2.6 Recommended Access Mechanism

There are multiple ways of accessing registers within the PCH. Here is the recommendation

	Access Method	Exception	Note
PCI(e) Configuration register	MMIO through PCIe* BAR (preferred), else CF8h/CFCh	1. ACPI device might have BAR1 for PCI CFG access 2. Config-disabled RP1/5/9 can use SBI programming method	PCIe* RP1/5/9 contains registers that affect other RP2/3/4/6/7/8...
Private Configuration register	MMIO through SBREG_BAR [1]	1. USB 2.0 AFE register must use SBI programming method 2. GPIO unlock register field must use SBI programming method 3. CIO2 FLIS register must use SBI programming method 4. ME0-based PCR should use the SBI programming method due to the FID requirement	

NOTE:

1. Due to the back to back write MMIO cycle restriction. Thus, all accesses that involve write have to be followed by a read. For GPIO and ITSS, back to back write is allowed after PCH early initialization (refer to section 2.2.2), but always do a read to any PCR before first write if unsure of the prior access (to other portid).

§ §

3 Flash Programming Considerations

Please refer to the latest PCH External Design Specification (EDS) and SPI programming Guide for details regarding the flash controllers.

3.1 Overview

The PCH supports two types of flash components: Firmware Hub (FWH) connected via the LPC bus or Serial Peripheral Interface (SPI) flash connected via the SPI bus. The following sections detail BIOS programming considerations for the flash component(s). Due to other system requirements, SPI should be the main interface used in this platform.

3.2 Boot BIOS and eSPI Strap

The Boot BIOS Strap (BBS) field and eSPI strap determines the destination of accesses to the BIOS memory range. The value of the BBS field can be read from the General Control and Status (GCS) register at PCR[DMI] + 274Ch[10] or SPI PCI offset DCh[6] or LPC/eSPI PCI offset DCh[6]. The BBS default values are determined by hardware straps. Please refer to the EDS for more information about the hardware straps.

Table 3-1: Boot BIOS Strap Definition

BBS[1] (0=SPI, 1=LPC/ eSPI)	eSPI Strap (0=LPC, 1=eSPI)	Boot Flash Location	Boot Flash Decode Range Location
0	0	Bios Boot from SPI	SPI PCI Config Register. Program SPI PCI-config (D31:F5) Bios Decode Range Configuration Registers. SPI uses its own Bios Decode Range to determine the Bios Boot Location
0	1	Bios Boot from SPI	
1	0	Bios Boot from LPC	LPC PCI Config Register. Program LPC PCI-config (D31:F0) Bios Decode Range Configuration Registers. LPC uses its own Bios Decode Range to determine the Bios Boot Location.
1	1	Bios Boot from eSPI	eSPI PCI Config Register Program eSPI PCI-config (D31:F0) Bios Decode Range Configuration Registers. eSPI uses its own Bios Decode Range to determine the Bios Boot Location

NOTE:
1. BBS has 3 locations (see above, from register programming interface angle)

BIOS Decode Enable is available from two locations based on the BIOS storage location. Location D31:F0 PCI offset D8h is used when BIOS is stored in eSPI/LPC.

Location D31:F5 PCI offset D8h is used when BIOS is stored in SPI. It also needs to be source decoded, refer to section 2.5.1.4.

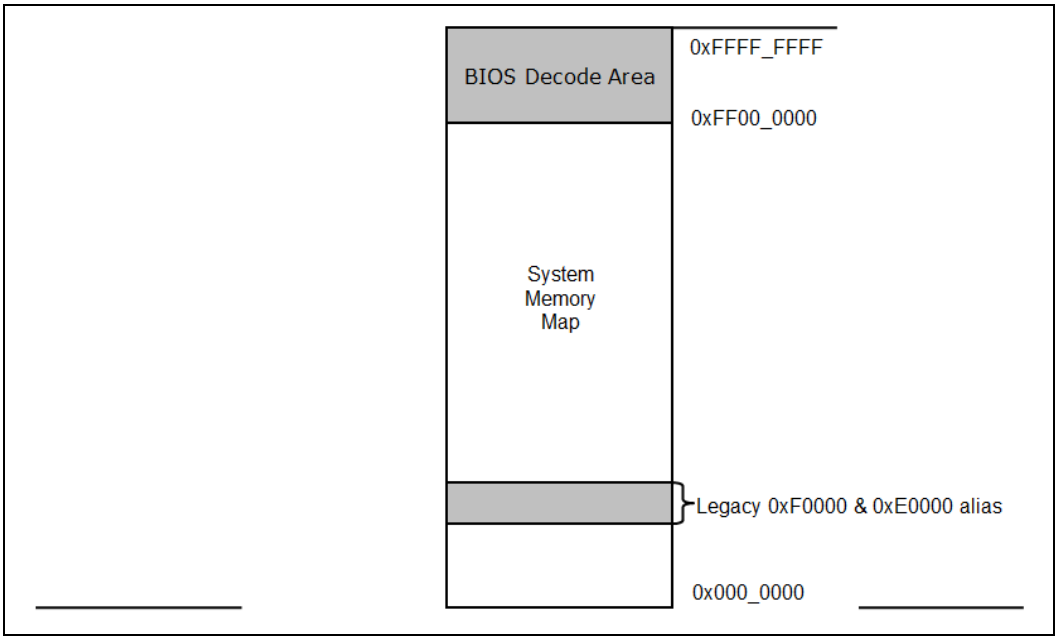
BBS is available in three locations as stated in earlier paragraph. All of them have to be in sync. The BBS in PCR[DMI] + 274Ch[10] is actually performing source decode to the SPI or LPC/eSPI. Software cannot overwrite BBS if the individual BIOS Interface Lock Down (3 of them, one for each) is set. If the BIOS Interface Lock Down bit is not set, writing to this field will take effect immediately. The BBS in PCR[DMI] is also no writable if "DMIC.SRL", PCR[DMI] + 2234h [31] is set.

3.3 Flash Memory Map

The System BIOS may be read by issuing memory reads to the BIOS decode area. The BIOS decode area is made up of 16-MB range of memory space decoded immediately below the 4-GB boundary from FF00_0000h – FFFF_FFFFh. The System BIOS can reduce the BIOS decode area by programming the BIOS Decode Enable register (LPC/eSPI PCI offset D8h for LPC/eSPI, SPI PCI offset D8h for SPI) with the exception of the region (FFF8_0000h – FFFF_FFFFh) and (FFB8_0000h – FFBF_FFFFh) which are always decoded. The BIOS Decode Enable register also can be used to control the decoding of the legacy F0000h and E0000h 64-KB segments, which can be used as an alias to the corresponding 64KB segments in the region below the 4-GB boundary, the decoding of these segments are enabled by default.

The following figure shows the BIOS decode areas.

Figure 3-1: Flash Memory Map



3.4 Flash Device Enumeration

The device node and BIOS Space range reserved depends on whether the BIOS supports a PnP or non-PnP system:

For a PnP BIOS, the BIOS must define a single device node that specifies the Register Space range for the device. The BIOS Space range is reserved through the Int 15h E820h function.

For non-PnP BIOS, the Intel FWH's ranges, both Register Space and BIOS Space, should be reserved through the Int 15h E820h function.

3.4.1 ACPI ASL Definition

For ACPI ASL definition please refer to the PCH reference code.

3.5 SPI Specific Configuration

Under host operation, the PCH SPI controller is only capable of operating in Descriptor mode. Descriptor mode is enabled by the presence of a Flash Descriptor in the first block of the first SPI flash component in the system. The PCH reads a descriptor signature in the flash descriptor during reset which will enable descriptor mode if the signature is determined to be valid. Please refer to the latest PCH EDS for details on the flash descriptor and functionality for Descriptor modes.

Softstrap definitions for Skylake Kaby Lake PCH can be found in the appropriate EDS and SPI Programming Guides.

Some SPI flash components have limitation on the number of times that status register can be changed, due to the implementation of their status register in its NVRAM. Because of this limitation, BIOS has to take precaution and prevent excessive changes to the status register, such as commands to lock/unlock SPI flash, etc.

Besides, the Quad Enable bit is often stored in the status register. BIOS must ensure that it does not change the state of this bit when writing to the status register.

3.5.1 SPI Configuration Registers

The SPI Host Interface registers are memory-mapped in its BAR0.

For Skylake PCH-LP/H B0 stepping and above, the SPI configuration space is to be hidden from OS by setting the SPI PCI offset DCh [9].

For a detailed description of these registers, please refer to the latest version of the PCH EDS.

3.5.2 SPI Read Configuration

The System BIOS can be read from SPI Flash device using Direct Read or Program Registers. The Boot BIOS Strap register (Refer to section 3.2) must be configured to SPI for BIOS boot using direct reads.

The System BIOS is encouraged to dynamically enable prefetching and caching only when the reads are sequential (for example, if shadowing the BIOS using consecutive DWord reads). When the prefetching and caching is enabled the PCH checks if the starting address for a read is aligned to the start of a 64-B block (i.e., address bits 5:0 are 00h). If this is not the case, then the PCH only reads the length specified by the

current read. If the read is aligned to the start of the 64-B block, then the read burst continues on the SPI pins until 64 bytes have been received.

The System BIOS must program SPI PCI offset DCh[3:2] = '10b' to enable SPI Flash prefetching and caching.

3.5.3 PCH SPI Initialization Flow

Below steps describes high-level BIOS enabling flow:

1. PCH detects and loads descriptor and SFDP tables.
11. Enable SPI Prefetching and caching.
12. Enable BIOS Flash protection. Refer to section 3.6
13. For fast booting, any power requirement on setting the "Clock and Power Gating Control" register, PCR[ESPISPI] + C004h, do it at End Of Pei.

3.5.4 Serial Flash Discoverable Parameter (SFDP)

The SFDP table is a data structure that SPI vendors would support in SPI flash device. The data structure provides SPI flash device information such as VSCC values and flash attributes. Please refer to the EDS document for details regarding SFDP usage.

Upon SPI reset the PCH SPI controller would detect the SFDP table presence in Chip Select#0 and Chip Select#1. The PCH SPI controller would use the SFDP table for all sequencing with the SPI flash cycle if SFDP table is discovered.

3.5.5 SPI Flash Operation Modes

The PCH SPI controller can communicate to the SPI flash device using three different methods:

1. Direct Read
2. Hardware Sequencing
3. Software Sequencing (host access using this method is no longer supported from Skylake PCH onwards)

3.5.5.1 Direct Read

The BIOS can access the flash controller by doing a direct read to the 16-MB address range just below the 4GB, FF00_0000h to FFFF_FFFFh. The BIOS Boot straps register must be configured such that BIOS resides on SPI. Please refer to section 3.2 above. In addition, the BIOS can reduce the size of the 16-MB address range by writing to the BIOS Decode Enable register (SPI PCI offset D8h). Please refer to section 3.3 above.

Direct write access to any region of SPI flash is not supported. Use Hardware sequencing methods for Erase and Write accesses.

3.5.5.2 Hardware Sequencing

The Hardware Sequencing uses Hardware Sequencing control and status registers for all sequencing with the SPI flash cycle. In Hardware Sequencing, the erase and the write instructions use the information in the SFDP table.

The PCH SPI controller would use the below instruction op-codes when Hardware Sequencing is used.

Table 3-2: Hardware Sequencer Instruction Set

Instruction Op-Code	Description
03h	Read Data
05h	Read Status
0Bh	Fast Read
06h	Write Enable
04h	Write Disable (Reserved, not used)
02h	Write Data / Program Data
01h	Write Status Register
3Bh / Discoverable	Dual Output Fast Read Discoverable opcodes are obtained from each component's SFDP table.
Discoverable	Dual I/O Fast Read
Discoverable	Quad Output Fast Read
Discoverable	Quad I/O Fast Read
Discoverable	4k, Erase Uses the value from SFDP (if available) else value from VSSC.EO register
Discoverable	64k, Erase
5Ah	Read SFDP, Use fast read timing with 8 wait states
9Fh	Read JEDEC ID

3.5.5.3 SPI Flash Operation Mode Recommendation

It is recommended that the BIOS use Hardware Sequencing exclusively for all SPI operation except direct read for BIOS region.

3.6 Flash Security Recommendation

Intel strongly recommends the following actions to help mitigate malicious software attacks to the BIOS region on the flash device.

BIOS needs to enable the BIOS Lock Enable (BLE) feature of the PCH by setting SPI PCI offset DCh[1] = 1b for SPI, or eSPI PCI offset DCh[1] = 1b for eSPI. When this bit is set, attempts to write the BIOS Write Enable (BIOSWE) bit in PCH will cause a SMI which will allow the BIOS to verify that the write is from a valid source.

BIOS needs to enable the “Enable in SMM.STS” (EISS) feature of the PCH by setting SPI PCI offset DCh[5] = 1b for SPI or setting eSPI PCI offset DCh[5] = 1b for eSPI. When this bit is set, the BIOS region is not writable until SMM sets the InSMM.STS bit,

to ensure BIOS can only be modified from SMM. Please refer to processor BWG for more details on InSMM.STS bit.

Note: If SPI/eSPI PCI offset DCh[11] is set, it allows SMI trigger on SPI write/erase to BIOS region when BIOSWE=0. The PCI offset DCh[11] provides no additional write protection but for the SPI write attempt logging. Thus, it is recommended not to enable this SMI if no logging is intended.

BIOS needs to enables SMI_LOCK (PMC PCI offset A0h[4] = 1b) which prevent writes to the Global SMI Enable bit (GLB_SMI_EN ABASE + 30h[0]). Enabling this bit will mitigate malicious software attempts to gain system management mode privileges.

BIOS also needs to set the BIOS Interface Lock Down bit in multiple locations (PCR[DMI] + 274Ch[0], LPC/eSPI PCI offset DCh[7] and SPI PCI offset DCh[7]). Setting these bits will prevent writes to the Top Swap bit (under their respective locations) and the Boot BIOS Straps. Enabling this bit will mitigate malicious software attempts to replace the system BIOS option ROM with its own code.

In PCH SPI controller the BIOS should set the Flash Configuration Lock-Down bit (SPI_BAR0 + 04[15]) at end of post. When set to 1, those Flash Program Registers that are locked down by this FLOCKDN bit cannot be written. Please refer to the EDS for which program registers are impacted.

In addition PCH SPI controller also offers five protected range registers that BIOS can enable:

- Protected Range 0 or FPR0 (SPI_BAR0 + 84h)
- Protected Range 1 or FPR1 (SPI_BAR0 + 88h)
- Protected Range 2 or FPR2 (SPI_BAR0 + 8Ch)
- Protected Range 3 or FPR3 (SPI_BAR0 + 90h)
- Protected Range 4 or FPR4 (SPI_BAR0 + 94h)

When enabled the PCH would block read or writes to protected BIOS ranges until next system reset.

Each of the protected range registers can be locked separately with corresponding bits in Discrete Lock Bits register[2] (SPI_BAR0 + 0Ch). Bit 8 of the register corresponds to FPR0, bit 9 to FPR1, up to bit 12 to FPR4. Each bit will be logical or'ed with FLOCKDN bit to determine if the corresponding register will be locked.

BIOS Flash Program Registers are now in SPI BAR0. The BIOS needs to make sure the SPI BAR0 content has not changed during runtime (SMM or non-SMM). It is not expected for this BAR to change because the SPI device is hidden from the OS for Skylake and Kaby Lake PCH LP/H B stepping and above (refer to section 3.5.1), but if it is ever different from the preallocated address, reassign it back. Refer to section 5.7 and 5.9.1.1 for the pre allocated range and reserved memory requirement.

For LPC/eSPI/SPI, its Top Swap is located at PCI offset DCh[4], is an RO field. These fields will reflect the PCR[RTC] + 3414h[0].

[2] DLOCK register is NOT applicable for Skylake PCH-LP A0, A1 and A2 steppings.

3.7 BIOS Region SMM Protection Enabling

If the following steps are implemented:

- Set the EISS bit (SPI PCI offset DCh [5]) = '1b'
- Follow the 1st recommendation in Section Flash Security Recommendation3.6

The BIOS Region can only be updated by following the steps below:

- Once all threads enter SMM
- Read memory location FED30880h OR with 00000001h, place the result in EAX, and write data to lower 32 bits of MSR 1FEh
- Clear SPI_SYNC_SS (SPI PCI offset DCh [8]) = '0' by write one clear (NOTE: must clear the SPI_SYNC_SS before changing WPD from 0 to 1).
- Set WPD bit (SPI PCI offset DCh [0]) = '1b'
- Modify BIOS Region
- Clear WPD bit (SPI PCI offset DCh [0]) = '0b'
- Read memory location FED30880h AND with FFFFFFFEh, place the result in EAX, and write data to lower 32 bits of MSR 1FEh.

Note: For platform using BIOS Guard, please see BIOS Guard BIOS Writer's Guide Section 2.1.2 for platform configuration and 2.3 for how to issue writes to BIOS Region.

3.8 PCH Soft Strap Access

From Skylake-PCH onwards, the flash descriptor observability is removed from the SPI controller. In order to access to the PCH soft strap information, this could be done by reading the descriptor region directly. The location of the soft straps is given by (16 * FLMAP1.FPSBA) + soft strap offset.

Note: For this proposed operation to work, the host must be given read access to the descriptor region.

Refer to the PCH SPI Programming Guide for the related soft strap offset.

§ §

4 Reset Control

4.1 Reset Control Overview

The reset vector starts program execution at physical address 0FFFFFFF0h. Since the processor cannot start program execution from this address in real mode, the first instruction at the reset vector is typically a far jump to a location in flash below 1 MB, in the 0E0000h to 0FFFFFh range.

Hard and soft resets are controlled by the Reset Control register (I/O Address CF9h.) This range is always enabled. See the PCH EDS for a description of I/O Address CF9h.

4.2 Hard Reset Control

The PCH generates PLTRST# when a hard reset is initiated through the Reset Control register (I/O Address CF9h.) To initiate a hard reset through the Reset Control register, software can:

- Write 06h to Reset Control register at CF9h (as known as platform Warm Reset) (SLP_S3#, SLP_S4#, SLP_S5# are not asserted), or
- Write 0Eh to Reset Control register at CF9h (as known as platform Cold Reset) (SLP_S3#, SLP_S4#, SLP_S5# are asserted).

4.3 CPU Only Reset Control

CPU only resets are controlled by the Reset Control register (I/O address CF9h). This I/O address is always enabled.

To initiate a CPU only reset, the following sequence should be followed:

1. Disable all bus master activity (clear Bus Master Enable bit 2 in all PCI device's configuration register Offset 04h)
2. Clear the SYS_RST field (bit 1) of the Reset Control Register at CF9h.
3. Initiate a CPU Only Reset by setting the RST_CPU field (bit 2) of CF9h.

A warm reset will partially initialize the processor without resetting the chipset. Therefore, the first instruction will be fetched from the flash at the reset vector address 0FFFFFFF0h. The jump at the reset vector will then jump to an address in shadowed RAM. The System BIOS is required to preserve the address jumped-to from the reset vector in shadow RAM and not to overwrite it for other uses.

4.3.1 Additional Concern of CPU Only Reset

There are 3 ways to issue a CPU only reset by system software or test tools.

1. Write 04h to Reset Control Register at CF9h.
2. Set bit0 to IO port 92h.

3. Write FEh to IO port 64h.

PCH still support those mechanisms, but CPU only reset is prohibited and not supported from CPU side. Therefore, system software and test tools should no longer issue those commands.

Note: To W/A the CPU only reset issue, system BIOS should promote the CPU only reset to CF9h 06h in the very beginning of reset flow. (It can be detected by checking the PCIEBAR is valid or not since PCIEBAR register goes not get reset in such case.)

4.4 Special Cycle Reset

The PCH will always generate a CPU Only Reset reset if the processor issues a "shutdown special cycle". There is no status bit for indicating a shutdown special cycle occurred and there are no disable bits to prevent the INIT from occurring.

4.5 Global Reset

There are two mechanisms to generate a Global Reset (resets both host and Intel ME), via a Global Reset MEI message or a CF9h IO write of 6h or Eh with "CF9h Global Reset (CF9GR)" bit set, PMC PCI offset ACh[20]. System BIOS should refer to Intel® Management Engine (ME) BIOS Writer's Guide to determine which Global Reset mechanism should be used.

4.6 SLP_S3#, SLP_S4#, SLP_SUS#, and SLP_A# Stretch Requirements

For PCH platforms, System BIOS needs to program the "SLP_S3# Minimum Assertion Width", "SLP_S4# Minimum Assertion Width" and "SLP_S3# Minimum Assertion enable", "SLP_S4# Assertion Stretch Enable", "Disable SLP Stretching After SUS Well Power Up", etc. appropriately in GEN_PMCON_B register and also SLP_SUS#/SLP_A# minimum assertion width in PM_CFG. Please consult with your board designer for optimal settings and refer to the relevant "PCH External Design Specification (EDS)" for details.

4.7 Processor Configuration Change Reset Requirement

If there are processor configuration changes, program the strap setting into "Set Strap Msg Data" register located at SPI_BAR0 Offset F8h [15:0] and follow the steps outlined in the "processor configuration reset flow" section of the Processor BIOS Writer's Guide and skip steps 3 and 4.

§ §

5 Basic BIOS Requirements

This section discusses the basic System BIOS requirements for a PCH-based platform.

5.1 Initializing Platform Controller Hub SBREG Base Address

PCH contains a memory-mapped chipset configuration space of 16MB, ”Configuration Registers”. From Skylake onwards, Configuration Register consumers own their own registers and are accessible via SB as Private Config Registers. Software has access to these via mechanisms in the P2SB as well, which mapped to a 16-MB range.

Earlier chipsets’ RCRB offset is no longer valid and need to be remapped to this new MMIO. Certain configurations (for example, DMI TC to VC mapping vs PSFs TC to IOSF channel mapping) are hosted at different sideband endpoints with different format.

BIOS is required to understand the registers differences and make sure all relevant configurations are updated. Before the registers can be accessed by software, System BIOS must initialize the P2SB SBREG_BAR register with a valid 64-bit, 16MB-aligned memory address, and then set the P2SB PCI offset 04h[1] MSE bit to 1 to enable this configuration space.

Once enabled, registers in this space can be accessed by byte, word, or DWord as long as the access does not cross a DWORD boundary. It is possible to first setup the base at below 4 GB, and move to above 4 GB later if chosen to. However, the ACPI ASL routines would not be able to access above 4 GB if 32 bit OS is used, thus it is better and easier to always map it under 4 GB if there is a possibility of running 32-bit OS.

The private configuration registers access are expected to in the early BIOS phase and thus the SBREG_BAR needs to be setup earlier. It’d be easier to preallocate an address range for it and use it through the boot including OS and SMM. P2SB can be hidden as per section below to prevent its BAR from being changed.

5.1.1 Hiding P2SB

P2SB device is to be hidden by the BIOS before PCI enumeration step by setting P2SB PCI offset E1h[0] to 1 using byte access (not to override E0h register value). BIOS can unhide it by clearing the same bit. After post, the unhiding should be done within SMM only to prevent OS from seeing it. Hiding of the P2SB device helps preventing BIOS/OS from enumerating the SBREG_BAR so that BIOS can continue to use hardcoded reference.

5.2 Initializing the I/O Base Registers

The PCH contains a number of registers accessed through the I/O blocks that are enabled by programming the I/O base address registers. The I/O base address register should be programmed before the I/O range is enabled, as described below. Failure to program the base address before enabling the I/O range will result in the address conflicting with devices in the 0-80h I/O range. The I/O base addresses

must be conflict free with other legacy I/O addresses and their aliases, such as the VGA I/O range at location 3B0h-3DFh and the alias at every following 1K-boundary (i.e., 7B0h-7DFh, etc.).

The I/O blocks that are located above offset 03Fh in PCI space that require System BIOS programming are listed in the table below.

Table 5-1: I/O blocks in the Intel PCH

Device Name	BAR Location	Size (Bytes)	Enable
ABASE	PMC PCI offset 40h	256	44h[7]
TCO	SMBUS PCI offset50h	32	50h[2]

NOTE:
1. It is foreseen that the ABASE and TCO addresses will be given a range that is under 1000h, to conserve more I/O space for the PCIe* RP. Example address: ABASE=D00h, TCOBASE=400h

5.3 Hiding and Function Disable Platform Controller Hub (PCH) Internal Devices

PCH provides a mechanism of hiding the PCI configuration space of the internal PCI devices. PCH also provides a mechanism to static disable a device. Devices that are not used in the system can be hidden or function disable before the System BIOS loads the OS to prevent the OS from erroneously loading drivers. Devices should be hidden or function disable before the System BIOS assigns system resources to prevent unused and unnecessary holes from being created in the system address space.

Certain device like P2SB can be hidden and unhidden by setting or clearing certain bit. Certain device like SATA, once hidden/disabled, can only be restored by a platform reset.

Be noted that the wording “function disable” used to be meant configuration disable in prior PCH generation, although sometime it also means disable both PCI configuration as well as the memory accesses, while having another register field that allow the configuration disable only. If have doubt, please refer to PCH EDS for details. This document will always refer the term “function disable” as complete hiding/disable of the primary interface (PCI configuration space, MMIO and IO), “configuration disable” or “hiding” as making BDF PCI configuration access returning UR.

For all devices that are intended to be hidden or function disabled permanently, there would be a corresponding bit field with the PSF register to be set as well.

NOTES:
1. Not all functions may be available on all SKUs. See EDS for details.
2. As said in section 2.5.2, it is possible to have IP default disable. BIOS needs to detect that (as example the XHCI), and disable the function in the PSF as well.

5.3.1 Fuse Disabled, Static Disabled, Non-Static Function Disable Device Handling

PMC provides a mean to read the fuse disable information for devices. For those devices that are fuse disabled or static PG'ed (Please refer to EDS for devices that could be static PG'ed), BIOS has to perform the PSF function disable step for the relevant device, by setting the relevant PSF's "AGNT_T0_SHDW_PCIEN" [device] Bit[8]. For Intel Serial I/O devices, only if all of its devices are having the static PG'ed bit set, then only the BIOS is required to perform the related PSF function disable programming.

If the BIOS policy for that device was to be enabled, but if it was static PG'ed, BIOS must perform a reset after BIOS reverse those static PG settings by clearing those bit(s) respectively in PWRMBASE + 620h and PWRMBASE + 624h. If the device is GbE, refer to section 10.2 for additional programming needs. For Intel Serial I/Odevices, reset is only needed if all devices were previously static PG'ed.

If the BIOS policy for that device was to be disabled and if it could be static PG'ed, the static PG programming action could be performed without a reset for boot time benefit. BIOS must perform additional power related programming, and perform the PSF function disable step for the relevant device as well.

For devices (except the PCIe* RP that is used for Intel® RST for PCIe* Storage remapping feature) that cannot be static PG'ed, the BIOS still need to perform the PMC function disable programming at NST_PG_FDIS_1, "PWRMBASE + 628h", beside other function disable step listed under each IP's corresponding section. There are IP that would function disable itself based on strap. For instance, the PCIe* RP# that can be configured to 4x1, 2x2, x2+2x1 and 1x4, the BIOS must also perform the PSF function disable step on those unused ports earlier by reading "Root Port Configuration Strap" of first root port of each controller. As it is not guaranteed that the first RP of a controller is not fused disabled, thus it is required to use sideband access for the "Root Port Configuration Strap" reading.

BIOS should also perform the PSF configuration hide on D20F4 earlier. BIOS should also perform the PSF function disable step on D30F7.

Prior to 3rd party code running, set "Static Function Disable Lock", PWRMBASE + 620h bit [31] to 1. And this must be done after all static PG and non-static PG programming done.

Note: Any PCI configuration read to the related BDF prior to the PSF steps above, could result in a system hang.

5.4 Platform Management Timers

The PCH power management timers are accurate to +-1 clock. This means the values 0 and 1 must not be programmed in a timer since it would result in an unpredictable count of –1 to +1.

This restriction applies to the Periodic SMI timer, the software_SMI timer, and the TCO timer (see Chapter 0).

5.4.1 Periodic SMI timer

PCH implements a Periodic SMI timer. The periodic SMI occurrence can be selected to have different granularity by programming the PER_SMI_SEL field, PMC PCI offset A0h[1:0].

APM System BIOS enables the Periodic SMI interrupt by setting the PERIODIC_EN bit, ABASE + 30h[14]. When the PERIODIC_STS bit, ABASE + 34h[14], is set, an SMI is generated. The SMI handler can check for system activity by reading the DEVTRAP_STS register, ABASE + 44h. If none of the system bits are set, the SMI handler can increment a software counter. When the counter reaches a certain number with no activity, the SMI handler can put the system into a lower power state. This feature should be disabled after switching to ACPI mode.

The software should clear the PERIODIC_STS bit by writing a '1' to it and set the PERIODIC_EN bit in order for the hardware to arm for the next periodic SMI . (Note that GBL_SMI_EN bit at ABASE + 30h[0] should also be set to 1 for periodic SMI to occur.)

5.4.2 Software SMI Timer

PCH implements a Software SMI timer. The Software SMI occurrence can be selected to have different granularity by programming the SWSMI_RATE_SEL field, PMC PCI offset A4h[7:6].

The status bit, SWSMI_TMR_STS, is located at ABASE + 34h[6] and the enable bit, SWSMI_TMR_EN, is located at ABASE + 30h[6]. (Note that Global SMI Enable bit at ABASE +30h[0] should also be set for SMIa to occur.) Software should clear the SWSMI_TMR_STS bit by setting it to a '1' and set the SWSMI_TMR_EN bit in order for the next software SMI to occur.

5.5 PCI/PnP Enumeration

The System BIOS is required to perform standard PCI/PnP enumeration and initialize all PCI/PCIe* devices/functions discovered. As part of this process, all PCI-to-PCI bridges must be initialized with secondary/subordinate bus numbers assigned and bridge windows programmed properly. In addition, I/O, memory, and interrupt resource allocation to PCIe* devices should be completed.

5.6 ACPI Support

The BIOS should include ACPI support, with proper ACPI tables built in memory and reported to OS.

ACPI BAR is located at PMC PCI offset 40h, with enable bit at PMC PCI offset 44h[7].

5.7 Reporting IO/Memory Resources to OS

Some of the memory and I/O address ranges used by PCH internal devices are assigned by the System BIOS using base address registers beyond the standard PCI header (above offset 40h), and thus is invisible to the OS. The System BIOS must report such memory/IO resource usage to the OS through standard software interfaces to avoid potential system resource conflict or WHQL test failures.

Table 5-2: BIOS BARs

Device Name	BAR Location	Size (Bytes)
ABASE	PMC PCI offset 40h	256
PWRMBASE	PMC PCI offset 48h	64K [2]
TCOBASE	SMBUS:50h	32
TBARB+TBARBH [1]	Thhermal:40h	4K

NOTES:
1. Do not recommend using TBARB. Refer to section 17.2
2. PWRMBASE is actually 4KB. But due to the source decode register usage, BIOS has to assume it is 64KB in size and 64KB aligned).

Equally true are those devices that are hidden but its memory/IO resource continued to be available for the BIOS/ACPI usage. ACPI devices' BAR is not listed here as they would be claim under relevant device scope. There are some fine-grain details of those BAR that might be still need to be declared under system resource, but the details are documented at its relevant section, such as section 22.7.1.

Table 5-3: Hidden Devices' BARs

Device Name	BAR Location	Size (Bytes)
SBREG_BAR [2]	P2SB PCI offset 10h	16M
TBAR+TBARH [1]	Thermal PCI offset 10h	4K

NOTES:
1. Depending on platform policy, the said device might not be hidden. It is expected the thermal device would not be hidden due to the DPTF POR.
2. If GPIO driver is to be used, then there is a hole within the SBREG_BAR range that will be claim by the GPIO driver. In this case, The SBREG_BAR memory range will be split into 3 ranges, with GPIO range claimed by the driver, and the rest being system resources.

Besides, there is also a special handling need for the SPI BAR0. Refer to section 3.6.

Table 5-4: SPI BAR

Device Name	BAR Location	Size (Bytes)
SPI BAR0	SPI PCI offset 10h	4K

The System BIOS should report the memory address ranges above via ACPI _CRS object as motherboard resources, with the device ID of PNP0C02.

The System BIOS should report all of the I/O register space used by the ACPI, TCO to the OS by using the PNP device node entries and by using the ACPI _CRS object.

They should be declared as Plug and Play motherboard resources with the Device ID of PNP0C02.

5.8 Cache Line Size Clarification

In accordance with PCI specification v2.3, System BIOS should program the number of cache lines, for PCI bus masters with memory write and invalidate, to the number of cache lines in the processor. For example, on Intel® Core™ 2 Duo processor-based systems, the cache line size for a bus master should be set to 64 bytes. The cache line size register, offset 0Ch, of a PCI device with bus master memory write and invalidate capability should initially be set to 10h since it accepts cache line size in terms of DWORDs. In the event a device does not support 64 bytes, it is then the responsibility of System BIOS to arbitrate between all devices implementing this register on the same bus to find the lowest common denominator.

Example:

On a given Intel Core 2 Duo processor-based system, there are two devices on the same PCI bus. Device A can only accept a cache line size of 32 Bytes. Device B can accept up to 128 Bytes. Since an Intel Core 2 Duo processor has a cache line size of 64 Bytes, System BIOS tries to program Device A and Device B to 64. When BIOS reads back the cache line size register on Device A to make sure it was programmed correctly, it sees that Device A did not accept the value of 64 Bytes. System BIOS next tries to program both devices with a cache line size of 32 Bytes. When reading back the register on Device A, BIOS sees that this was programmed correctly. Next, System BIOS checks to ensure Device B was programmed correctly. Since Device B can accept up to 128 Bytes, it also accepts the value of 32 Bytes.

Results: Processor cache line size 64 Bytes

Device A cache line size 32 Bytes

Device B cache line size 32 Bytes

5.9 Intel Reserved Memory Address Space

On PCH, memory address space from FED2_0000h through FED7_FFFFh is reserved. System BIOS must mark this memory address space range as reserved when reporting system resources to the OS via the use of PnP Device Node, E820 table, and ACPI _CRS object to prevent OS from using this memory address space range.

5.9.1 Special PCH BIOS Reserved Memory Allocation

There are quite some memories ranges need to be pre-allocated in PCH. Such as SBREG_BAR, etc. There are also potential needs to allocate unique memory ranges to the devices' MSI-X table BAR and PBA table BAR, with unknown BAR size and such programming is needed in early BIOS phase, PCH BIOS component will reserve certain memory range as reserved range for BIOS usage. For Skylake and Kaby Lake, this range will be from 0FD000000h till FE7FFFFFh, a total of 24 MB and it is guaranteed that the Skylake compatible processor will not be using this range.

These range is further partitioned into the below usages

Size (MB)	Start	End	Usages
16	FD00 0000	FDFF FFFF	SBREG_BAR
8	FE00 0000	FE7F FFFF	Other BAR (static) allocation usages

5.9.1.1 Detailed Recommended Static Allocation

Size	Start	End	Usages	Retain Address After BIOS PCI Enum
16 MB	FD000000	FDFFFFFF	SBREG_BAR	Yes, device hidden
64 KB	FE000000	FE00FFFF	PMC MBAR	Yes, non-standard BAR
4 KB	FE010000	FE010FFF	SPI BAR0	No. But during runtime, the same address is used.
88 KB	FE020000	FE035FFF	Intel Serial I/O devices BARs when operate in ACPI Mode	Yes if ACPI Mode– device hidden, No for PCI mode
8 KB	FE036000	FE037FFF	SDIO when operate in ACPI Mode	Yes if ACPI Mode– device hidden, No for PCI mode
4 KB	FE03C000	FE03CFFF	Thermal Device in ACPI mode (non POR)	Yes if ACPI Mode– device hidden, No for PCI mode
256 KB	FE0C0000	FE0FFFFF	Intel TH FW BAR	Yes, non-standard BAR If Intel TH is enabled, the MSE need to be remained set. See below method for achieving this.
1 MB	FE100000	FE1FFFFF	Intel TH MTB BAR	Yes, for non-FSP build, hide Intel TH right after init completion, unhide through PCI enum callback

Size	Start	End	Usages	Retain Address After BIOS PCI Enum
2 MB	FE200000	FE3FFFFF	Intel TH SW BAR	Yes, for non-FSP build, hide Intel TH right after init completion, unhide through PCI enum callback
64 KB	FE400000	FE40FFFF	CIO2 when operate in ACPI mode (non POR)	Yes if ACPI Mode– device hidden, No for PCI mode
2 MB	FE600000	FE7FFFFF	Temp Address – does not have requirement to be reserved, but PCH RC is free to use this range for temporary address need.	

SPI BAR0 (4 KB) is free to use any empty range within this memory range or somewhere else, as long as it is reserved from the OS usage. Refer to section 5.7

5.9.1.2 Intel® RST for PCIe* Storage Remapping Feature Memory Allocation Support

During the remapping initialization, there might a need to allocate unique address to the PCIe* storage device MSI-X capability related BARs (cross reference to section 24.3) in which the OS guarantee not to allocate it to the PCH SATA ABAR or it could cause address collision within the PCIe storage device itself. The way this could be achieved is by allocating those addresses from the reserved memory range. Thus, the memory allocator is free to use the address from FD000000h till FDFFFFFFh for the purpose.

Note: There is no address collision with other BAR usages as stated in section 5.9.1.1 because the address assigned to the remapped PCIe* storage device is hidden from the rest.

5.10 R/WO Registers

Some of the registers in PCH are implemented as Read/Write-Once (R/WO) while they are defined as Read-Only by the PCI Express* specification. It is required that System BIOS must write to such registers once during POST and before booting to OS, so that the registers will become Read-Only.

Please refer to EDS for complete register details.

5.11 Platform Controller Hub (PCH) Boot Checklist

This section lists the minimum steps necessary to boot using the PCH.

1. Initialize SBREG base address register and enable it (P2SB PCI offset 10h~17h). Refer to section 5.9.1 for the assigned address.
2. Before any interrupt is enabled, set ITSS.MMC, PCR[ITSS] + 3334h [0] to 1.
3. Initialize the LPC and/or FWH/SPI decode/enable registers according to platform design (LPC/eSPI or SPI PCI offset 80h-83h, D0h-DCh).
4. Program Port 80h cycle routing (LPC/eSPI or PCIe*) so that BIOS post codes can be displayed properly.
5. Program PCH GPIO multiplexing and level settings.
6. Check RTC Battery failure bit or RTC checksum failure.
7. Program I/O Base addresses (ABASE, TCOBASE, etc).
8. Disable the Watchdog timer expiration from causing a system reset by using 'GPP_B_18' hardware strap instead of setting No Reboot, PCR[SMB] + 0Ch[1]. This "No Reboot" does not disable the watchdog timer, just the watchdog timer's effect of causing a system reboot on timer expiration. Steps need to be added to prevent unwanted TCO messages from occurring due to the third expiration of the TCO timer.
9. Hide unused PCI functions by referring to section 5.3.
10. Check Y2K rollover TCOBASE + 04h[7] (NEWCENTURY_STS bit) and increment century if necessary.
11. Program SATA PCI offset 90h (MAP) to configure serial ATA controllers.
12. Program Sub-System and Sub-Vendor IDs for the internal devices in PCH.
13. Relocate SMBASE, clear SMI status bits at ABASE + 34h, and enable SMIs
14. If I/OxAPIC is being used in the system, it can be enabled by setting the APIC Enable bit, P2SB PCI offset 64h[8].
15. Determine if the system is resuming or this is a normal boot by following steps 2 and 3 in section 19.11.1.

5.12 Intel® Stable Image Platform Program (SIPP)

For platforms supporting Intel® SIPP, System BIOS will need to enable the Compatibility Revision ID (CRID) feature by:

1. Write the value 1Dh to the RevID field in PMC PCI offset 08h

Notice that prior to this write, the RevID field will report the Stepping Revision ID (SRID). The first write to this register will lock it from further modifications.

Reference Code relies on the ability to read the SRID value from the RID register in order to initialize the PCH correctly, thus it requires that BIOS does not write to the RID register prior to execution of the reference code.

Refer to the latest External Designed Specification for more information regarding Intel® SIPP and enabling CRID.

Note: It is always good programming practice to enable CRIDs late in BIOS POST as there could be other BIOS code that relies on SRID for workarounds/sightings/errata.

5.13 BIOS Guide on Using RTC RAM

The following is the guidance for BIOS on storing secure / sensitive data in RTC RAM:

1. For credential security/confidentiality ensure only trusted code is executing while credential is stored in RTC RAM; trusted code must scrub RTC credential prior to exit.
2. For Data integrity protection, set RTC Memory locks (Upper 128 Byte Lock and Lower 128 Byte Lock) at PCR[RTC] + 3400h[4] and PCR[RTC] +3400h[3].

Note: Once locked bytes 38h – 3Fh in each of the Upper and Lower Byte blocks, respectively, cannot be unlocked until next reset.

5.14 Additional Programming Steps

5.14.1 Processor Set Strap Message

Once the strap message programming is done, set “Set_Strap_Lock”, SPI_BAR0 + F0h [0] to 1b.

5.14.2 EFI Tick Rate

BIOS can change EFI tick rate to 1ms for CSM free OSes.

5.14.3 PSF Register Access

After BIOS send EOP message to CSE, the BIOS will lose access most PSF register ranges, which cover access to registers such as AGENT_T0_SHADOW, etc. except those above B000h. It is important to have those needed programming done prior to that.

5.14.4 PSF EOI Multicast Configuration

BIOS will always program the PSF EOI multicast to forward it to P2SB (ITSS). Any IOAPIC controller found attached directly to the PCIe* RP, the BIOS would clear the related “EOI Forwarding Disable” bit in the PCIe* RP PCI configuration and the RP would be added into the target.

For ITSS in Skylake PCH-LP only:

1. Program PCR[PSF_1] + 4060h to 38A00h
2. Program PCR[PSF_1] + 4050h [1, 0] to [1, 1]
3. Program PCR[PSF_3] + 4054h = 38A00h
4. Program PCR[PSF_3] + 404Ch[1,0] to [1, 1]

For ITSS in Skylake PCH-H only:

1. Program PCR[PSF_1] + 4058h = 38B00h
2. Program PCR[PSF_1] + 4048h[1,0] to [1, 1]
3. Program PCR[PSF_3] + 4054h = 38B00h

4. Program PCR[PSF_3] + 404Ch[1,0] to [1, 1]

For ITSS in Kaby Lake PCH-H only:

1. Program PCR[PSF_1] + 405Ch to 38B00h
2. Program PCR[PSF_1] + 404Ch [1, 0] to [1, 1]
3. Program PCR[PSF_3] + 4054h = 38B00h
4. Program PCR[PSF_3] + 404Ch[1,0] to [1, 1]

For Skylake and Kaby Lake PCH-LP:

RP	Destination ID	RP	Destination ID
1	18100h	7	18302h
2	18101h	8	18303h
3	18102h	9	18500h
4	18103h	10	18501h
5	18300h	11	18502h
6	18301h	12	18503h

For Skylake and Kaby Lake PCH-H:

RP	Destination ID	RP	Destination ID
1	18000h	11	18302h
2	18001h	12	18303h
3	18002h	13	18500h
4	18003h	14	18501h
5	18100h	15	18502h
6	18101h	16	18503h
7	18102h	17	18700h
8	18103h	18	18701h
9	18300h	19	18702h
10	18301h	20	18703h

5.14.5 VTd Support Requirement

For system that supports VTd, beside the requirements listed in the Skylake and Kaby Lake processor BWG, the BIOS also needs to

1. Set PCR[PSF_1] + 4010h [1, 0] to [1, 1]
2. Set PCR[PSF_2] + 4010h [1, 0] to [1, 1]
3. Set PCR[PSF_3] + 4010h [1, 0] to [1, 1]
4. Set PCR[PSF_4] + 4010h [1, 0] to [1, 1]

5.14.6 Additional Power Management Configuration

BIOS must follow below setting to enable ITSS dynamic clock gating.

1. Program PCR [ITSS] + 3300h [4:0] = 11011b

5.15 Legacy IO low latency mode support

BIOS must follow below setting to enable low latency mode for legacy IO, which is required for some robotic applications.

1. Program P2SB PCI offset E4h[5] = 0b.
2. Program P2SB PCI offset E8h[0] = 1b
3. Program P2SB PCI offset E8h[1] = 1b
4. Program P2SB PCI offset EAh[0] = 1b
5. Program P2SB PCI offset EAh[1] = 1b
6. Program PCR [ITSS] + 3300h [0] = 0b
7. Program PCR [ITSS] + 3300h [1] = 0b
8. Program PCR [ITSS] + 3300h [2] = 0b
9. Program PCR [ITSS] + 3300h [3] = 0b
10. Program PCR [ITSS] + 3300h [4] = 0b
11. (*)Program PCR [DMI] + 21A8h [1:0] = 00b
12. (*)Program CPU DMIBAR.LCTL offset 88h[1:0] = 00b

Note(*): Step 11 and 12 are only required for PCH-H

§ §

6 Interrupt Support on Platform Controller Hub (PCH)

6.1 PCI IRQ Routing on Platform Controller Hub

The OS has an ability to dynamically route PCI interrupts to interrupt requests (IRQs). Usually in the System BIOS a SETUP option is implemented to decide whether a PnP OS is going to be booted. The System BIOS must assign IRQs to all PCI devices during power-on self-test (POST) when the PnP OS Option is set to "Absent." System BIOS must assign IRQs only for the boot devices, and not "all PCI devices" when a PnP OS that is IRQ routing capable is "Present" These interrupts, once chosen, can be moved if the OS finds there is a conflict with the current usage of the interrupt.

In order to support assignment and reassignment of PCI IRQs, the OS will need to know how the system board has wired each motherboard integrated PCI device and each PCI slots interrupt pins to the PCI Interrupt Router's interrupt pins. This chapter describes the capabilities PCH has to allow customization of such routings, and how System BIOS should report this platform-specific information to the OS.

6.1.1 PCI* Interrupt to IRQ Router

There are 8 PIRQ pins supported in PCH, named PIRQ[A# :H#], that route PCI interrupts to IRQ's of the 8259 PIC. The PCH also connects the 8 PIRQ[A# : H#] to 8 individual IOxAPIC input pins, as shown in Table 6-1 below.

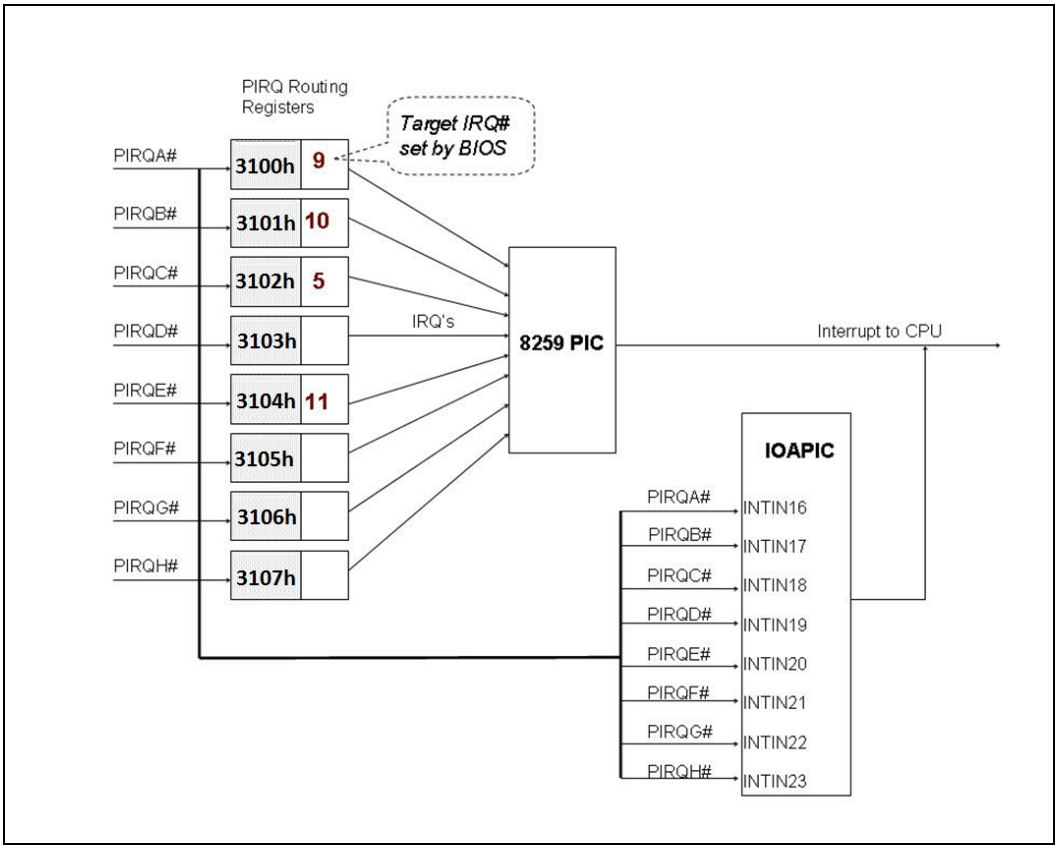
Table 6-1: Platform Controller Hub (PCH) PIRQ Routing Table

PIRQ# Pin	Interrupt Router Register for PIC	Connected to IOxAPIC Pin
PIRQA#	PCR[ITSS] + 3100h	INTIN16
PIRQB#	PCR[ITSS] + 3101h	INTIN17
PIRQC#	PCR[ITSS] + 3102h	INTIN18
PIRQD#	PCR[ITSS] + 3103h	INTIN19
PIRQE#	PCR[ITSS] + 3104h	INTIN20
PIRQF#	PCR[ITSS] + 3105h	INTIN21
PIRQG#	PCR[ITSS] + 3106h	INTIN22
PIRQH#	PCR[ITSS] + 3107h	INTIN23

Older PCH Operation Region Coding	Skylake/Kaby Lake PCH Operation Region Declaration

// LPC Bridge - Device 31, Function 0
// Define the needed LPC registers used by ASL.
 	
scope(_SB){
OperationRegion(_SB.PCI0.LPCB.LPC1, PCI_Config, 0x00, 0x100)
Field(_SB.PCI0.LPCB.LPC1, AnyAcc, NoLock, Preserve)
{
Offset(0x02),
CDID,16,
Offset(0x08),
CRID, 8,
Offset(0x60),
PARC, 8,
PBRC, 8,
PCRC, 8,
PDRC, 8,
Offset(0x68),
PERC, 8,
PFRC, 8,
PGRC, 8,
PHRC, 8,
Offset(0xAC),
, 8,
, 8,
XUSB, 1
}
	// ITSS
	// Define the needed ITSS registers used by ASL on Interrupt
	// mapping.
	External (\SBIT) //SBREG_BAR + (ITSS_PORT << 16)
	
	scope(_SB){
	OperationRegion(ITSS, SystemMemory, \SBIT + 0x3100, 0x8)
	Field(ITSS, ByteAcc, NoLock, Preserve)
	{
	PARC, 8,
	PBRC, 8,
	PCRC, 8,
	PDRC, 8,
	PERC, 8,
	PFRC, 8,
	PGRC, 8,
	PHRC, 8,
	}

Figure 6-1: Platform Controller Hub (PCH) PIRQ to IRQ Router



6.1.2 Interrupt Routing for Platform Controller Hub (PCH) Internal PCI Devices

PCH provides maximum flexibility of interrupt routing for internal PCI functions by allowing System BIOS to configure which INTx pin is used by an internal PCI function to drive its interrupt request, and which input pin (PIRQy) 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). From Skylake PCH onwards, there are certain devices such as Intel Serial I/O, SCS, ISH and OTG devices, they are capable of mapping its own INTx to IRQ# directly. This allows the platform design to reduce the possibility of interrupt sharing among these PCI functions. Figure 6-2 below shows a logical illustration of this programmable interrupt routing control. It should be noted that INTx pin, when used in the PCI Express* context, represents a virtual wire rather than a physical signal pin (PCI Express* delivers interrupts by in-band messages instead of side-band pins).

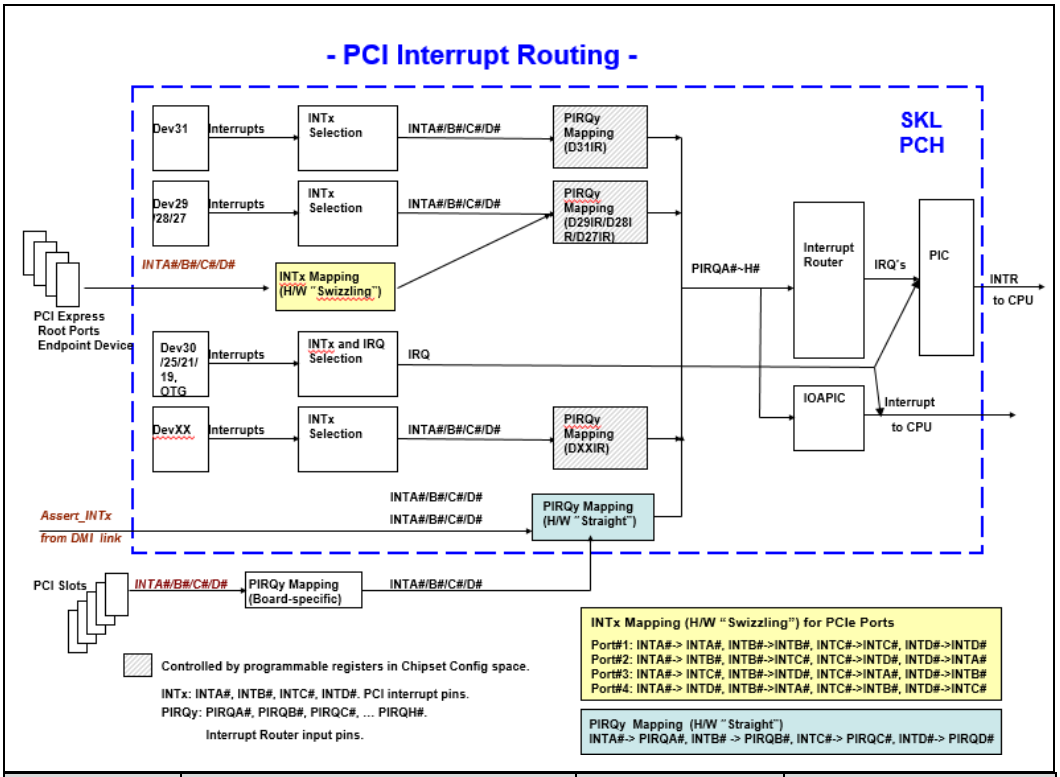
The descriptions of the interrupt routing control registers as well as their default values, which are located from PCR[ITSS] + 3100h onwards, can be found in PCH EDS. Note that in Skylake and Kaby Lake PCH, there is radical change in interrupt assignment, inclusive of INTx assigning for the internal PCI function. Please refer to EDS and subsequence paragraphs for details.

When using these registers to implement a PCI interrupt routing scheme, it is important to remember the following rules defined in the PCI Specification:

The rule of thumb is:

1. For a single-function PCI device capable of generating an interrupt, use INTA.
2. For a multi-function device, at least one of the interrupt-capable functions must use INTA.
3. 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. (Example: PCI IRQ routing table for all internal PCI devices and PCI slots, MPS table, ACPI _PRT objects, etc.).

Figure 6-2: Platform Controller Hub (PCH) / PCI Interrupt Routing Control



			I²C
			I²C
			I²C
			I²C
			I²C
			I²C

6.1.2.1 Interrupt Assignment Rules

There are devices such as Intel Serial I/O, SCS, ISH and OTG devices; its INTx is directly mapped to IRQ within its own control, without the ITSS.PIRx PIRQy mapping.

If the IRQ# chosen is >=24, it must be unique (not share with others). If the IRQ# chosen is <=23, it can be anything, except:

- 0, 2, 8,
- Same as IRQ# used by LPC SERIRQ,
- Same as IRQ# used by devices with the same sideband port ID

Note: For Intel Serial I/O and SCS devices setup as PCI mode under device 30, if all are using IRQ#>=24, only up to 4 devices under the device 30 can be enabled at a given platform, and device with function number 0 always being enabled. If the IRQ# chosen is <=23, then it is possible for the device with different sideband port ID to share same IRQ# (and same INTx at the same time), means between Intel Serial I/O devices and SCS devices.

For the Intel Serial I/O, SCS, ISH and OTG device, the BIOS must further set the related bit in IPC0-3 (PCR[ITSS] + 3200h – 320Ch) for the corresponding IRQ chosen to make it active low, regardless of being PCI mode or ACPI mode.

For INTx assignment, there are devices assign its INTx by writing to a private configuration space. For some other devices, their INTx field could be a RWO field, in which it must be written once to assign the given INTx. Or their field could be RW with a lock elsewhere. There is a case (Intel TH) whereby the field is RO only.

Note: It is a requirement for the platform BIOS to make sure all PCI devices offset 3Dh becomes RO prior to OS. It means all RWO field if any, must be written once, all RW field if any, must be locked.

6.1.3 Interrupt Routing for PCI Express* Root Ports

The PCI Express* root ports on PCH deserve more interrupt routing discussion than other devices.

Interrupts generated by the root ports themselves (for example, PME or Hot Plug event interrupts) are governed by the same rules and configuration registers as those for all other internal PCI Express* functions described above. On the other hand, the root ports play the role of forwarding interrupts received from their secondary interfaces to the upstream root complex. In an effort of minimizing the possibility of interrupt sharing, a different interrupt mapping, or “swizzling”, is defined for these root ports when forwarding interrupts upstream on behalf of downstream PCI Express* devices at their secondary interfaces, i.e., at the downstream end of the PCI Express* links. Table 6-2 shows this interrupt mapping for downstream devices.

Table 6-2: Platform Controller Hub / PCI Express* Root Port Interrupt Mapping for Downstream Devices

Port	INTA#	INTB#	INTC#	INTD#
1	INTA#	INTB#	INTC#	INTD#
2	INTB#	INTC#	INTD#	INTA#
3	INTC#	INTD#	INTA#	INTB#
4	INTD#	INTA#	INTB#	INTC#
5	INTA#	INTB#	INTC#	INTD#

Port	INTA#	INTB#	INTC#	INTD#
6	INTB#	INTC#	INTD#	INTA#
7	INTC#	INTD#	INTA#	INTB#
8	INTD#	INTA#	INTB#	INTC#
9	INTA#	INTB#	INTC#	INTD#
10	INTB#	INTC#	INTD#	INTA#
11	INTC#	INTD#	INTA#	INTB#
12	INTD#	INTA#	INTB#	INTC#
13	INTA#	INTB#	INTC#	INTD#
14	INTB#	INTC#	INTD#	INTA#
15	INTC#	INTD#	INTA#	INTB#
16	INTD#	INTA#	INTB#	INTC#
17	INTA#	INTB#	INTC#	INTD#
18	INTB#	INTC#	INTD#	INTA#
19	INTC#	INTD#	INTA#	INTB#
20	INTD#	INTA#	INTB#	INTC#
21*	INTA#	INTB#	INTC#	INTD#
22*	INTB#	INTC#	INTD#	INTA#
23*	INTC#	INTD#	INTA#	INTB#
24*	INTD#	INTA#	INTB#	INTC#

Note: Skylake/Kaby Lake PCH-LP and Skylake/Kaby Lake PCH-H have different number of ports.

The **INTx** in boldface on the top row in the table are interrupts received from the downstream devices (before the swizzling). The rest of the INTy in the table are interrupts forwarded upward by the root ports (after the swizzling). This swizzling is hard-wired in PCH, and is not programmable. As an example, when a downstream device at root port 2 generates INTA#, then the root port will convert this INTA# into INTB# before forwarding it upstream.

From Skylake PCH onwards, the swizzled interrupt from the endpoint device is further subject to the INTx to PIRQy mapping through the ITSS D29IR, D28IR and D27IR. To maintain the legacy direct mapping (INTA to PIRQA etc.), it is recommended not to change the default value within these registers.

System BIOS is required to handle this swizzling and report this routing information to the OS in a similar way in which it handles PCI Express* slots on the PCI Express* bus.

6.1.4 Interrupt Routing for DMI Assert_INTx Messages

Interrupt events from the processor can be sent down the DMI link to PCH in the form of Assert_INTx TLP messages. Conceptually these messages represent PCI INTx

virtual wires fed into PCH, and are internally routed (mapped) straight to PIRQ[A/B/C/D] as shown in Table 6-3:

Table 6-3: Intel PCH DMI Assert_INTx Interrupt Mapping Scheme

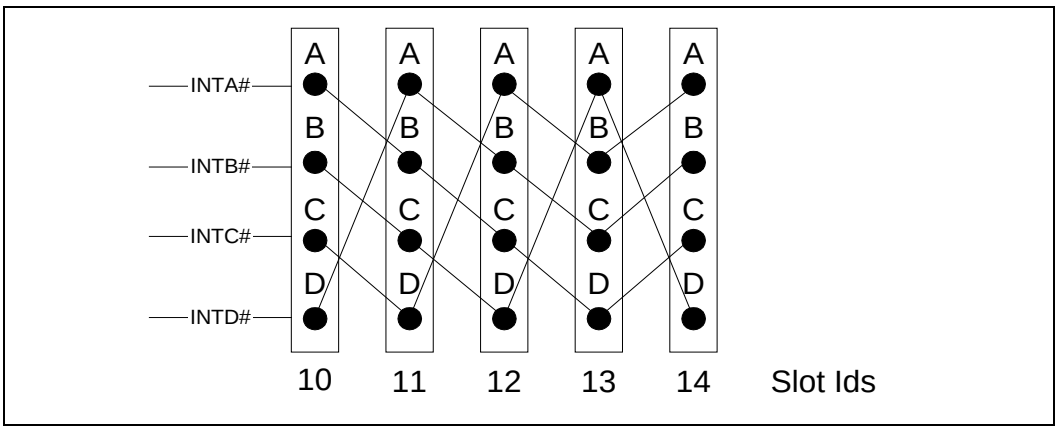
DMI Message Received by Intel PCH	Mapped to
Assert_INTA	PIRQA#
Assert_INTB	PIRQB#
Assert_INTC	PIRQC#
Assert_INTD	PIRQD#

6.1.5 Interrupt Routing for PCI Expansion Slots

Each PCI system board consists of one or more PCI expansion slots. Each slot has four interrupt pins, known as INTA#, INTB#, INTC#, and INTD#. The PCI Interrupt Router has several input pins, known as PIRQA#, PIRQB#, PIRQC#, ... PIRQH#. The INTx# pins for each slot may be wire OR'd with other INTx# pins from the same or other slots, and these groups of pins may also be connected to a PIRQy# pin on the Interrupt Router.

PCI expansion slots on the system motherboard should always be described based on the 'wire routing' of the slot(s). Since many PCI adapter cards are single function and must use INT#A, motherboard designers design the PCI expansion slots so that the interrupts lines leading from the slot are rerouted so as to minimize the occurrence of interrupt sharing. This is sometimes referred to as 'swizzling', Figure 6-3 illustrates this technique:

Figure 6-3: PCI Expansion Slot Interrupt Routing



6.1.5.1 Example for PCI Slot Interrupt Routing

If two single function adapter cards (X and Y) were plugged into slots 10 and 11 respectively, it can be observed from Figure 6-3 that the function on card 'X' would continue to use INTA#. However, the function on device 'Y' would actually be routed to use INTD#. Likewise, if device 'X' were plugged into slot 14, its function INTA# pin would get routed to INTC#. Note that interrupt routing also works for adapter cards that are multi-function capable.

Since a card plugged into a slot may use any one of the INTx pins, the System BIOS must provide an interrupt assignment entry for all interrupt signals (INTA# through INTD#) for each expansion slot on the system board.

6.1.6 Reporting Interrupt Routing to OS

The PCI interrupt routing scheme for all motherboard and chipset internal devices, PCI expansion slots and PCI slots behind the root ports is platform specific and needs to be reported to the OS by System BIOS via the following interfaces:

- PCI BIOS INT1Ah Service, Sub-function B1xxh (PCI BIOS 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)

Please refer to the above specifications for detailed formats of interrupt routing tables and data structures.

Note: "$PIR" PCI IRQ Routing Table (Microsoft specification, non-ACPI environment) can no longer be supported due to hardware change.

6.1.6.1 Example PRT Packages for Interrupt Routing

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

```
//-----------------------------------------------------------------------------------------
// PCH INTERRUPT ROUTING _PRT EXAMPLE (PIC MODE)
//-----------------------------------------------------------------------------------------

Scope(\_SB) {

    Device(PCI0) {            // PCI root 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(){

        // Device 31 (SATA, SMBus)
        Package(){0x001FFFFF, 0, LNKD, 0 },
        Package(){0x001FFFFF, 1, LNKB, 0 },

        // Device 29 (USB)
        Package(){0x001DFFFF, 0, LNKH, 0 },
        Package(){0x001DFFFF, 1, LNKD, 0 },
        Package(){0x001DFFFF, 2, LNKC, 0 },
        Package(){0x001DFFFF, 3, LNKA, 0 },

        // Device 28 (PCH PCI Express Root Ports)
        Package(){0x001CFFFF, 0, LNKE, 0 },
        Package(){0x001CFFFF, 1, LNKF, 0 },
        Package(){0x001CFFFF, 2, LNKG, 0 },
        Package(){0x001CFFFF, 3, LNKH, 0 },

        // Device 27 (High Definition Audio)
        Package(){0x001BFFFF, 0, LNKA, 0 },

        // Device 25 (GbE)
        Package(){0x0019FFFF, 0, LNKA, 0 },
```

```
// Device 1 (Processor PCI Express graphics port)
Package(){0x0001FFFF, 0, LNKA, 0 },

// Device 2 (Processor Internal Graphic)
Package(){0x0002FFFF, 0, LNKA, 0 },
}) // end _PRT


    Device(PCI1) {            // DMI to PCI Bridge
    Name(_ADR, 0x001E0000)        // Device 30
    Name(_PRT, Package(){
    // PCH LAN Controller
    Package(){0x0008FFFF, 0, LNKE, 0 },
    // 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 PCI1


    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

                Device(PEX5) {
                Name(_ADR, 0x001C0004)  // PCI Express Root Port5
                Name(_PRT, Package(){
                // PCI Express Slot5
                Package(){0x0000FFFF, 0, LNKA, 0 },
                Package(){0x0000FFFF, 1, LNKB, 0 },
                Package(){0x0000FFFF, 2, LNKC, 0 },
                Package(){0x0000FFFF, 3, LNKD, 0 },
                })
                } // end PEX5

                Device(PEX6) {
                Name(_ADR, 0x001C0005)  // PCI Express Root Port6
                Name(_PRT, Package(){
                // PCI Express Slot6
                Package(){0x0000FFFF, 0, LNKB, 0 },
                Package(){0x0000FFFF, 1, LNKC, 0 },
                Package(){0x0000FFFF, 2, LNKD, 0 },
                Package(){0x0000FFFF, 3, LNKA, 0 },
                })
                } // end PEX6

                Device(PEX7) {
                Name(_ADR, 0x001C0006)  // PCI Express Root Port7
                Name(_PRT, Package(){
                // PCI Express Slot7
                Package(){0x0000FFFF, 0, LNKC, 0 },
                Package(){0x0000FFFF, 1, LNKD, 0 },
                Package(){0x0000FFFF, 2, LNKA, 0 },
                Package(){0x0000FFFF, 3, LNKB, 0 },

                })
                } // end PEX7

                Device(PEX8) {
                Name(_ADR, 0x001C0007)  // PCI Express Root Port8
                Name(_PRT, Package(){
                // PCI Express Slot8
                Package(){0x0000FFFF, 0, LNKD, 0 },
                Package(){0x0000FFFF, 1, LNKA, 0 },
                Package(){0x0000FFFF, 2, LNKB, 0 },
                Package(){0x0000FFFF, 3, LNKC, 0 },
                })
                } // end PEX8


            } // end PCI0

        } // end _SB scope
```

6.2 ACPI Device IRQ Routing on Platform Controller Hub

6.2.1 Interrupt Routing for Internal ACPI Devices

There are multiple devices within PCH can be configured in ACPI mode, and each of them could be assigned a different IRQ. It is important to the same IRQ assignment rules as stated in section 0, inclusive of the GPIO IOAPIC if it is being used.

The devices fall under this are Intel Serial I/O devices, SCS devices, ISH.

Note: Driver requirements need to be taking into account prior configuring devices to ACPI mode or PCI mode.

6.2.2 Interrupt Routing for External Devices through GPI

PCH GPIO provides 2 IRQ interrupt routing mechanism, one of them is by configuring the GPIO into GPIO Driver mode by:

1. Configure the GPIO pin into GPI by setting ["Pad Mode", "GPIO RX Disable", "GPIO TX Disable"], PADCFG0[11:10, 9, 8] to [00b, 0, 1].
    a. If the pin was previously GPO, need an intermediate step whereby ["GPIO RX Disable", "GPIO TX Disable"] are transitioned t0 [1, 1] before being [0, 1]
2. Set related bit in "Host Software Pad Ownership"[x] to 1b,
3. Set "GPIO Driver IRQ Route" to choose either IRQ14 or IRQ15 and declare the same interrupt resource within the GPIO ACPI device

Note: This has a dependency on the OS GPIO driver interrupt handling capability

Another way is to route the GPI to APIC by:

1. Configure the GPIO pin into GPI by setting ["Pad Mode", "GPIO RX Disable", "GPIO TX Disable"], PADCFG0[11:10, 9, 8] to [00b, 0, 1].
    a. If the pin was previously GPO, need an intermediate step whereby ["GPIO RX Disable", "GPIO TX Disable"] are transitioned t0 [1, 1] before being [0, 1].
2. Set "GPIO Input Route IOxAPIC", PADCFG0[20] to 1b,
3. Declare the related interrupt resource ("Interrupt Select" PADCFG1[7:0]") within the ACPI device that uses it.
4. Refer to section 6.7.2.1 for additional steps needed to enable IOxAPIC[119:24]

Note: It is important to make sure there is no interrupt sharing with the internal devices. In another word, the internal device's IRQ assignment should not be the same as the one used by the GPI IOxAPIC interrupt.

Note: Besides, if 2 GPIO are mapped to same IRQ#, then only one of them could be used for interrupt.

6.2.2.1 GPIO Interrupt Pin Isolation

If the external devices that are driving the GPI for interrupt, is power gated, it could stop driving (float) or assert the pin. There would be a need to reconfigure the GPIO before that to prevent spurious interrupt and reduce leakage power.

For I/O Apic level interrupts:

1. Disable GPIO event detection with RxEvCfg = 2h
2. If the interrupt is active low, override Rx internal pad state to 1b, by setting RxRaw1 = 1. The RxRaw1 can be part of the init flow if the interrupt mode is known.
3. Disable GPIO Rx input buffer with GPIORxDis = 1b
4. Power gate the external device

For GPIO Driver Interrupt:

1. Disable GPIO event detection with RxEvCfg = 2h.
2. Disable GPIO Rx input buffer with GPIORxDis = 1

3. Power gate the external device

When the power is reapplied to the device, the above sequence is to be executed in reverse order.

6.3 Serial IRQs

The only System BIOS requirement to use IRQs as a serial IRQ is to enable the function in LPC PCI offset 64h[7] and select continuous or quiet mode, LPC PCI offset 64h[6]. The serial IRQs are OR'ed with the IRQs generated internally in PCH. IRQ 0, 8, 13, and 16-and-above (APIC) are not supported by serial IRQs.

PCH requires that the System BIOS first set the SERIRQ logic to continuous mode operation for at least one frame before switching it into quiet mode operation. This operation should be performed during the normal boot sequence as well as a resume from STR (S3).

Note: You can set Serial IRQ in quiet mode for better power savings.

6.4 PIRQ Enable/Disable Handling by Platform Controller Hub (PCH)

PCH supports Serial IRQ and a PIRQ scheme. PCH supports a message for 21 serial interrupts which includes the 12 ISA interrupts (IRQ0-1, IRQ3-7, IRQ9-12 and IRQ14-15), SMI#, IOCHK#, PCI INTA# - INTD# but excludes IRQ0, IRQ8 and IRQ13 from the SERIRQ stream as they are generated internally.

Interrupts routed using the PIRQ Routing registers in 8259 mode are shared with SERIRQ. When PIRQ routing is enabled (PIRQ[n]_ROUT register(s) bit 7=0), the SERIRQ interrupt slot for the interrupt programmed into (PIRQ[n]_ROUT register(s) bits[3:0]) will be ignored by PCH.

- When PIRQ routing is disabled (PIRQ[n]_ROUT register(s) bit 7=1), the PCH interrupt controller will start sampling the SERIRQ slot for that interrupt. Since SERIRQ slot is "high," the interrupt controller sees the "high" on the SERIRQ slot and recognizes this as the request for an interrupt; hence a spurious interrupt is generated.
- PCH will generate multiple spurious interrupt(s) if all the following conditions are met:
    - IRQx is unmasked in the 8259 (IRQ programmed into the PIRQ register)
    - IRQx is configured as level triggered in 8259 (PCH sampling active high assertions in this case)
    - PIRQ routing register pertaining to IRQx changed from "enable" to "disable"
    - The spurious interrupt is continuously serviced by the interrupt handler.

If the PIRQ routing is disabled, software should first mask the 8259 from generating a spurious interrupt or be able to handle the extra interrupt generated during the

transition. This is the normal sequence that the OS uses while changing interrupts for PCI devices and switching from 8259 to IOAPIC usage.

6.5 NMI Handling in Platform Controller Hub (PCH)

6.5.1 PCH Settings to Generate NMI

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

1. PCH SERR# assertion
5. IOCHK signal (Serial IRQ bus only)
6. NMI message from the Processor.
7. NMI_NOW bit in the TCO1_CNT register

Software reads the NMI Status and Control Register (I/O port 61h) register to determine the NMI source. After the NMI interrupt routine processes the interrupt, software clears the NMI status bits by setting the corresponding enable/disable bit to '1'. See the PCH EDS for the complete definition of the NMI Status and Control Register.

The NMI Enable bit in the NMI Enable / RTC index register (I/O port 70h, bit 7) can be used to mask the NMI signal and disable/enable all NMI sources. This is useful when initializing ECC memory or before an NMI handler has been installed. The TCO logic can be programmed to make the NMI first generate a SMI. The SMI handler can then log the memory error and generate a NMI by writing the NMI_NOW bit.

See the PCH EDS for the complete definition of the NMI Status and Control Register as well as the NMI Enable Register.

6.5.2 NMI Handling in an OS Environment

PCH is capable of generating NMI's from various sources. Many of these NMI sources are currently not handled by the OS environment. The following NMI's generated by PCH due to the sources listed below have to be kept disabled or in case the System BIOS enables these NMI sources these events should be handled in the SMI handler of the System BIOS by routing the NMI to generate an SMI. The routing of NMI to SMI is possible by setting the NMI2SMI_EN bit in the TCO1_CNT register (TCOBASE + 08h[9]).

This is a list of some of the NMI sources that have to be handled in the SMI handler of the System BIOS.

Table 6-4: Typical NMI Sources

NMI Sources To Be Handled In SMI Handler Of System BIOS
Parity Error Response Enable (PCIE RP PCI offset 3Eh[0]).
DMISERR_STS bit (TCOBASE + 04h[12]). The enable bit for the DMISERR_STS events is present in the processor and memory controller of the system (for example, SERR# on multiple bit errors).

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

Note: The status of NMI2SMI_STS (TCOBASE + 04h[0]) won't reflect the TCO_STS of PMC SMI_STS.

6.5.3 Steps for Handling NMI in Platform Controller Hub (PCH) Systems

The System BIOS should execute the minimum steps described below for handling an NMI in PCH-based systems:

Steps for execution:

1. Save the contents of I/O Register CF8h.
2. Save the current RTC Index by reading port 74h.
3. Disable the NMI generation by setting port 70h[7] = 1b in.
4. Clear the PERR# and SERR# status bits in PCI Register Offset 06h-07h for all PCI Slots in the system.
5. Clear the SERR# causing status bits in the memory controller (for example: the single bit error flag and multiple bit error flag registers will be cleared in this step).
6. Clear the SERR# and NMI status bits in the TCO Register space.
7. Clear the PERR# and SERR# status bits in the Processor and PCH (Device#31, all functions) at PCI Register Offset 06h-07h.
8. Clear the Primary and Secondary PERR# and SERR# status bits in Device #30.
9. Clear the SERR# status bit in port 61h.
10. Restore NMI Disable bit in port 70h.
11. Restore RTC Index
12. Restore the contents of I/O Register CF8h.
13. Exit from the NMI handler.

Sample Code: (Base on older platform, register offset, etc would not be reflecting Skylake or Kaby Lake PCH)

```
Cmos_addr_port          equu      070h
Sound_port              equ       061h
PCI_REGISTER_INDEX      equ       0cf8h
MKF_PM_BASE_ADDRESS     equ       0400h
;memory location that keeps copy of NMI mask bit (i.e., bit7 last written
to port 70h)
NMI_Mask_Save                 extern                near ; Bit7 = saved
NMI mask bit

int_02  proc     near
                 push    si
                 push    bx
                 push    eax
```

```
                    push    edx
;; Step #1
                    mov     edx, PCI_REGISTER_INDEX
                    in      eax, dx
                    push    eax
;; Step #2
                    in      al,74h
                    or      al,80h                        ; disable NMI
;; Step #3
                    out     cmos_addr_port,al
                    push    ax                                ; save RTC index
bit[6:0]

                    xor     bx, bx
.586p
                    push    edx
                    push    eax
                    xor     si, si

;; Step #4
;; This code will try to clear the PCI Slots SERR# and PERR## status bits
(Bus1)
loop_for_clearing_pci_slot_status:
                    mov     edx, 0cf8h
                    mov     eax, 080010004h
                    mov     ah, byte ptr cs:PCISlotEquatesTbl[si]
                    out     dx, eax
                    mov     dx, 0cfch
                    in      eax, dx
                    out     dx, eax
                    inc     si
                    cmp     byte ptr cs:PCISlotEquatesTbl[si], 0FFh
                    jne     loop_for_clearing_pci_slot_status

;; Step #5
;; Clear the Processor Error status bits
                    mov     edx, 0cf8h
                    mov     eax, 0800000c8h
                    out     dx, eax
                    mov     dx, 0cfch
                    in      eax, dx
                    out     dx, eax;This clears Processor SBE and MBE flags
in C8h

;; Step #6
;; Clear the Status bits in TCO register space
                    mov     dx, MKF_PM_BASE_ADDRESS
                    add     dx, 064h    ; get to TCO + 04h(ABASE + 64h)
                    in      ax, dx
                    and     ax, 01800h ; clear only bit12 and bit11
                                       ; Processor SERR_STS
                                       ; and Processor NMI_STS
                    out     dx, ax

;; Step #7
;; This code will try to clear the Processor/PCH SERR# and PERR## status
bits (Bus0)
                    xor     si, si
                    loop_for_clearing_IMC_pch_status:
                    mov     edx, 0cf8h
```

```
                    mov     eax, 080000004h
                    mov     ah, byte ptr cs:IMCPCHDevicesEquatesTbl[si]
                    out     dx, eax
                    mov     dx, 0cfch
                    in      eax, dx
                    out     dx, eax
                    inc     si
                    cmp     byte ptr cs:IMCPCHDevicesEquatesTbl[si], 0FFh
                    jne     loop_for_clearing_imc_pch_status

;; Step #8
;;This code will try to clear the PCH Device#30 SERR status bits
;;Start adding this additional code for disabling and clearing the parity
response enable
                    mov     edx, 0cf8h
                    mov     eax, 08000F01Ch
                    out     dx, eax
                    mov     dx, 0cfch
                    in      eax, dx                 ; register 1e-1fh in PCH
                                                    ; Dev#30 is interesting
                    test    eax, 0C1000000h         ; DPED, SSE, and DPE
                    jz      no_status_bits_set
                    or      bx, 001h
                    no_status_bits_set:
                    out     dx, eax; clear all the status bits in Reg1Eh-1Fh

                    mov     edx, 0cf8h
                    mov     eax, 08000F004h
                    out     dx, eax
                    mov     dx, 0cfch
                    in      eax, dx  ; register 06-07h in PCH Dev#30 cleared
                    out     dx, eax

;; Step #9
;; Clear the port 61h based SERR# status bits
;; This code is not really enabling the IOCHK based NMI's
                    in      al, sound_port
                    push    ax
                    and     al, 0FBh  ; make bit2 =0
                    out     sound_port, al
                    in      al, sound_port
                    test    al, 0C0h  ; Is bit 7, 6  set to a 1??
                    jz      no_status_bit_port61
                    or      bx, 002h
                    no_status_bit_port61:
                    in      al, sound_port
                    or      al, 004h; set bit2 again to a 1 to clear bit7
                    out     sound_port, al
                    pop     ax
                    out     sound_port, al

                    pop     eax
                    pop     edx

;; Check out the setup option that indicates whether the NMI message
;; needs to be displayed on screen
                    mov      al, Q_NMI_DISP
                    out     070h, al
                    in      al, 071h
                    test     al, Q_NMI_DISP_MASK
```

```
                jnz     dont_display_nmi_message

                test    bl, 003h
                jz      dont_display_nmi_message

;; Path for displaying the NMI message
;; Add code here to display the NMI message to the user

dont_display_nmi_message:

;; Step #10
                pop     ax
                and     al,7fh    ; al = saved RTC index
                mov     ah, byte ptr NMI_Mask_Save
                and     ah, 80h
                or      al, ah    ; restore NMI mask bit


;; Step #11
                out      70h,al                   ; restore port 70h
;; Step #12
                pop     eax
                mov     edx, PCI_REGISTER_INDEX
                out     dx, eax
                pop     edx
                pop     eax
                pop     bx
                pop     si
;; Step #13
                iret

int_02  endp

;; these are the equates for the PCI Slots and devices whose SERR# and
;; PERR# status bits should be cleared
;; these are device number shifted left by 3 and OR'ed with function
;; number in lower three bits
PCISlotEquatesTbl:
        db      000h    ; Device #0
        db      008h    ; Device #1
        db      010h    ; Device #2
        db      018h    ; Device #3
        db      020h    ; Device #4
        db      030h    ; Device #6
        db      040h    ; Device #8
        db      0FFh    ; end of table

IMCPCHDevicesEquatesTbl:
        db      000h    ; Device#0
        db      008h    ; Device#1
        db      0D0h    ; Device#26, Function 0
        db      0D1h    ; Device#26, Function 1
        db      0D2h    ; Device#26, Function 2
        db      0D3h    ; Device#26, Function 3
        db      0D7h    ; Device#26, Function 7
        db      0D8h    ; Device#27, Function 0
        db      0E0h    ; Device#28, Function 0
        db      0E1h    ; Device#28, Function 1
        db      0E2h    ; Device#28, Function 2
        db      0E3h    ; Device#28, Function 3
```

```
db        0E4h    ; Device#28, Function 4
db        0E5h    ; Device#28, Function 5
db        0E6h    ; Device#28, Function 6
db        0E8h    ; Device#29, Function 0
db        0E9h    ; Device#29, Function 1
db        0EAh    ; Device#29, Function 2
db        0EBh    ; Device#29, Function 3
db        0ECh    ; Device#29, Function 4
db        0EFh    ; Device#29, Function 7
db        0F0h    ; Device#30, Function 0
db        0F2h    ; Device#30, Function 2
db        0F3h    ; Device#30, Function 3
db        0F8h    ; Device#31, Function 0
db        0F9h    ; Device#31, Function 1
db        0FAh    ; Device#31, Function 2
db        0FBh    ; Device#31, Function 3
db        0FFh    ; End of Table
```

6.5.4 Routing NMI to SMI

System BIOS may enable an NMI in case of multiple-bit ECC errors. In case of single-bit ECC errors, it may be necessary to prevent an NMI from happening as these types of single-bit memory errors can be scrubbed and corrected by the chipset. The System BIOS can log an entry in the SMI handler detailing the number of single-bit errors that have happened in the system. In a single-bit error case, the System BIOS can prevent an NMI from happening by routing an NMI to the SMI handler by setting the NMI2SMI_EN bit in the TCO1_CNT Register, TCOBASE + 08h[9]. The System BIOS should program the PCH TCO logic to generate an SMI. For specific details on enabling/handling TCO logic SMI# see Chapter 0.

The following example is being given for a PCH-based system where the System BIOS has already configured the Processor to generate a SERR# to PCH on a correctable single-bit ECC error or an uncorrectable multiple-bit error.

In this configuration, the Processor will generate an SERR# message when a single-bit or multiple-bit ECC error happens in the system. When PCH TCO logic receives the SERR# message, an NMI will be generated. This NMI can be routed to an SMI if the NMI2SMI_EN bit in PCH (TCO1_CNT Register, TCOBASE + 08h[9]) is set. In addition the System BIOS should program the PCH TCO logic to generate an SMI by set the TCO_EN bit (SMI_EN Register, ABASE + 30h[13]). The System BIOS should add the following functionality to the SMI# handler:

1. Read the source of the SMI# was TCO by checking if the TCO_STS bit in the SMI_STS Register, ABASE + 34h[13] = 1b. If this bit is not set, then continue with normal SMI# processing and do not execute any of the steps below.
2. Read the TCO source was Processor SERR by checking if DMISERR_STS bit (TCOBASE + 04h[12] = 1b). If this bit is not set, then continue with normal TCO SMI# processing and do not execute any of the steps below.
3. If Processor SERR was caused by a single-bit ECC error, then log the occurrence of this single-bit error and clear the DMISERR_STS bit by setting TCOBASE + 04h[12]. Continue with step 13.
4. If the Processor SERR was due to a multiple-bit error, then continue with step 5.
5. Generate an NMI by doing the following steps:
6. Read the NMI2SMI_EN bit (TCOBASE + 08h[9]), save it for future restore.

7. Clear the NMI2SMI_EN bit (TCOBASE + 08h[9]).
8. Enable NMI by setting the NMI_EN bit (IO port 070h[7]) = 0b.

Note: IO port 070h status must be restored prior to exit from SMI.

9. Set the NMI_NOW bit (TCOBASE + 08h[8]) = 1b.
10. Clear the NMI_NOW bit. To do this, write a 1b to NMI_NOW (TCOBASE + 08h[8]).
11. Restore the NMI2SMI_EN bit (TCOBASE + 08h[9]) saved in step 6.
12. Clear the DMISERR_STS bit by setting TCOBASE + 04h[12].
13. Continue with TCO SMI# processing.

6.6 SMI Handling in Platform Controller Hub (PCH)

6.6.1 PCH Settings to Generate SMI

Refer to PCH EDS for details.

6.6.2 SMI LOCK Bit

When the SMI_LOCK bit, PMC PCI offset A0[4], is set, writes to the GLB_SMI_EN bit (ABASE + 30h[0]) for enabling and disabling SMI generation have no effect. The SMI_Lock bit can only be cleared by a PLTRST#.

6.6.3 Setting the EOS Bit

Setting the EOS bit, ABASE + 30h[1], will de-assert the SMI# 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.

6.6.4 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.

6.7 Processor Interrupt Support

This section is moved from “Processor Interface” Chapter. BIOS developer can refer to system programmer’s manual and processor BWG for more details. The System BIOS boot flow is described follows:

1. The System BIOS will configure the PCH IOxAPIC for Processor Interrupt delivery by setting the registers as described in the previous section.
2. The System BIOS will build the MP and/or ACPI APIC tables for the OS.
3. Control is passed to the boot strap loader to handle control to the OS.

4. OS will switch the interrupt operation from PIC or virtual wire mode to symmetric I/O Mode (IOAPIC usage).
5. When the OS switches the mode of interrupt delivery to the IOxAPIC the interrupts are delivered using the PSB Interrupt mechanism.

6.7.1 Dual/Quad Processor Considerations

PCH does not support dual processor clock control. The ACPI System BIOS must only report support for the C1 and C2 states for dual processor designs unless other external hardware solutions are implemented. C2 is not supported by Quad Core processor design. Such solutions would have to account for the different processor blocks and stop grant cycles and are beyond the scope of this document.

In a dual processor system, the software may place the 2nd processor in the INIT IPI state, which causes the processor to enter a loop, which waits for another IPI (INIT IPI or a STARTUP IPI). In this loop where the processor is waiting for an IPI, the processor will not respond to FLUSH, A20M, INTR, INIT, NMI, PREQ and SMI.

While in this wait for IPI state, the processor may not respond with a Stop-Grant cycle.

6.7.2 Processor Interrupt Delivery Mechanism

The PCH supports the delivery of interrupts to the processor by using one of the two mechanisms:

1. Interrupt delivery by using the INTR/NMI VLM to the processor, when PCH IOxAPIC is disabled.
2. Interrupt delivery from the IOxAPIC by sending interrupt messages through address remapping which is based in the Bus: Device: Function field associated with the request when PCH IOxAPIC is enabled.

6.7.2.1 Configuration of the Platform Controller Hub (PCH) IOxAPIC

The requirements for System BIOS to enable IOxAPIC are:

1. Set the AE bit, P2SB PCI offset 64h[8] = 1b.
2. Build the MP table and/or ACPI APIC table for the OS.
3. Maximum Redirection Entries (MRE) in APIC Version Register (VER), offset 01h, [23:16] has to be written once for Microsoft Windows* OS.

System BIOS may select the IOxAPIC range using the APIC Range Select field, P2SB PCI offset 64h[7:0]. This field defines address bits 19:12 of the IOxAPIC range.

In addition Skylake/Kaby Lake PCH supports IOAPIC entries 24-119. The requirements for System BIOS to enable IOAPIC entries 24-120 IOxAPIC are:

1. Set the MAX_IRQ_ENTRY_SIZE field at PCR[ITSS] + 31FCh[15:9] to 0 for 120 entry size, to 1 for 24 entry size.
2. Set the Maximum Redirection Entries (MRE) in APIC Version Register (VER), offset 01h, [23:16] to 77h for 120 entry size, 17h for 24 entry size.

6.7.2.1.1 Support for a IOxAPIC Attached to a PCI Express* Port

To support an IOxAPIC behind an PCH PCI Express* Root Port, the System BIOS needs to set the following bits:

Dxx:Fn:D8h[7] = 1b, where Dxx:Fn is the Function that the IOxAPIC device is attached to.

6.7.2.2 Steps Involved In Delivering the Interrupt

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

1. PCH detects that an interrupt event has happened and sets the IRR bit in the IOAPIC associated with that interrupt
2. PCH 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 PCH follows step#3 and #4.
3. PCH automatically flushes all the upstream buffers
4. PCH delivers the interrupt to the processor 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”.
5. After this 32-bit memory write cycle is received by the Processor SA, the write occurs to the processor.

Table 6-5: 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 6-6: 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)	

§ §

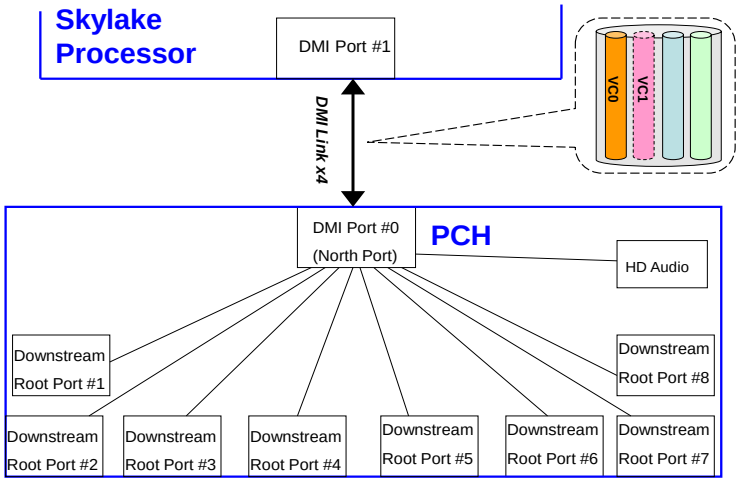
7 Direct Media Interface (DMI) and On-Package Direct Media Interface (OP-DMI)

Direct Media Interface (DMI) is the inter-connection between the Processor and the Skylake/Kaby Lake H and Skylake/Kaby Lake -LP Halo PCH components of the Intel chipset. It has similar facilities as PCI Express* (such as virtual channels). DMI consists of two internal ports, one on each side of the link. The PCH side of the DMI port is also called "North Port".

The reader is expected to be familiar with PCI Express* Specification 2.0 and related terminology (Endpoints, Root port, Isochrony, ASPM etc.).

Figure 7-1 below shows a conceptual diagram of DMI.

Figure 7-1: Platform Controller Hub (PCH) DMI Link and Root Ports



- PCH DMI and Root Ports -

OP-DMI refers to the On-Package Interface (OPI)-based DMI. Skylake/Kaby Lake -LP is the PCH die included in a MCP. OP-DMI provides the die-to-die connection between the processor complex and Skylake/Kaby Lake -LP.

7.1 DMI Link Speed and Training

Skylake onwards, DMI Gen 3 speed (8GT/s) is supported. The piece of BIOS code needed to train the DMI link to 8.0 GT/s data rate needs to be fully fetched from the boot block region and must reside in memory, prior to starting the Link Equalization so that processor can execute the BIOS code from memory. The DMI link will not be

accessible when link equalization is in progress. Recommendation on the execution location would be provided once characterization is done.

7.1.1 Gen 2/3 Link Speed Configuration

Default value of DMI Link Speed is GEN1. BIOS in order to change to Gen 2 or Gen 3 needs to check supported speeds in LCAP.MLS (PCR[DMI] + 21A4h[3:0]) register and set proper speed value in LCTL2.TLS (PCR[DMI] + 21B0[3:0]). After configuration BIOS is required to perform DMI Link Training and wait for completing it. As DMI is an interconnect between SA and PCH speed configuration needs to be done at both ends – please refer to SA BIOS Writer's Guide for details.

For Gen 3 Training, refer to section 7.2.4 for further initialization need before Gen3 training. On training started, BIOS will poll for the "Equalization Complete" field on the processor side and must not touch any register on the PCH side before completion.

7.2 Platform Controller Hub (PCH) DMI/OPIO Configuration

The configuration registers for the PCH DMI/OPIO port reside in the private configuration space and resemble the capability structures of a standard PCI Express* root port. For this reason, the software configuration of the PCH DMI/OPIO port described in this section (Virtual Channels, ASPM, etc.) is similar to that of a standard PCI Express* root port.

7.2.1 DMI Virtual Channel

PCH DMI supports 3 standard virtual channels (VC0 and VC1 and VCm(VC7)) as defined in the PCI Express* Specification, and uses a fixed arbitration policy among these virtual channels. VCm is the highest priority; VC0 is the lowest priority. The virtual channels, on the DMI link are transparent to the OS and operate under the rules in the PCI Express* Specification.

VC0 is always enabled as the default virtual channel on DMI for all traffic classes (TCs). Software is not able to disable VC0, although it can modify the TC/VC mapping on VC0. Unless re-configured by software, all transactions on DMI initiated by all internal and external devices travel through VC0.

For VC1 to be operational on DMI, it has to be configured and enabled at both ends of the DMI link (i.e., Processor and PCH). VC1 must only be enabled on DMI when Intel® High Definition Audio is enabled on the platform. See Section 9.4.1 for details on configuring VC1 on DMI when Intel® High Definition Audio is enabled.

Table 7-1: Recommended TC/VC Mapping

Traffic Class (TC)	Recommended TC[3:0] Encoding	Virtual Channel (VC)	Recommended VCID[3:0] Encoding	Recommended TC/VC Map [7:0]
TC0	0000b - 0h	VC0	0000b - 0h (hardwired)	0000_0001b - 01h
TC1	0001b - 1h	VC1	0001b - 1h	0000_0010b - 02h

Traffic Class (TC)	Recommended TC[3:0] Encoding	Virtual Channel (VC)	Recommended VCID[3:0] Encoding	Recommended TC/VC Map [7:0]
TCM	0111b - 7h	VCM	0111b – 7h	1000_0000b - 80h

7.2.2 DMI Link Active State Power Management (ASPM)

For PCH platforms, it is recommended that L1 ASPM link states be enabled and L0s must be disabled on DMI for Skylake/Kaby Lake -H/S platform. See section 8.3.1 for ASPM requirements on PCH platforms.

7.2.3 PCH Display Port

System BIOS is required to program following registers so that the PCH display logic are properly decode the Register Access that are used to communicate with the North Display in the Processor

1. Set P2SB PCI offset A0h[15:0]= 0010h

7.2.4 Additional PCH DMI and OP-DMI Programming Steps

System BIOS must perform the following steps (in the order listed) to configure the PCH DMI port on both cold boot (G3/S5) and S3/S4 boot flows. Besides that, please refer to the SA BIOS Writer's Guide for correct DMI register programming in details.

1. System BIOS must program below before any upstream bus master cycles are enabled.
   a. Set PCR[DMI] + 2234[17:16] = 11b (using byte access)
2. System BIOS must program below before using any PCH functionality.
   a. Set PCR[DMI] + 2088h = 00109000h (DWord) – Set error mask for UES register
   b. Set PCR[DMI] + 20ACh[30] = 1b– Unsupported transactions on DMI will not cause setting UES.URE bit
   c. PCR[DMI] + 2300h[1] = 1b (for PCH With OP-DMI ONLY)
   d. PCR[DMI] + 2348h[3:2] =00b (for PCH With DMI ONLY)
   e. PCR[DMI] + 2348h[1:0] = 00b (for PCH With DMI ONLY)
3. This must be done when DMI is either Gen1 or Gen2, after that link can be trained to Gen 3 (for PCH With DMI ONLY)

   Set PCR[DMI] + 21BCh[27:24] = 7h

   Set PCR[DMI] + 21BCh[11:8] = 7h

   Set PCR[DMI] + 21C0h[27:24] = 7h

   Set PCR[DMI] + 21C0h[11:8] = 7h
4. This must be done when DMI is either Gen1 or Gen2, after that link can be trained to Gen 3 (for PCH with DMI ONLY).

   a. Set PCR[DMI] + 2450h[16] = 1b

   b. Set PCR[DMI] + 2338[9] = 1b

5. Configure DMI Link Speed (for PCH With DMI ONLY)

   a. Please refer to the System Agent BIOS Writer's Guide on Supported Link Speed field in Link Capabilities register in processor complex.

   b. If target link speed is GEN2, it can be configured as early as possible. Else if the target link speed is GEN3, refer to section 7.1 for further programming requirement

   c. Set PCR[DMI] + 21B0h[3:0] = AND(Supported Link Speed in processor complex, PCR[DMI] + 21A4h[3:0])

      - Please refer to the System Agent BIOS Writer's Guide to perform DMI Link Retrain after configures new DMI Link Speed.

6. System BIOS must ensure the same DMI TC/VC mapping is programmed at both ends of the DMI link. In order to ensure correct programming of Virtual Channels in the PCH, variable is defined to share with System Agent as below and BIOS needs to set those the following registers before MRC execution:

Configure TC/VC mapping. Refer to Table 7-1.

   a. PCR[DMI] + 2014h[7:0] = 01h – TC0->VC0

   b. PCR[DMI] + 2020h = 81000002h (DWord) – TC1->VC1

   c. PCR[DMI] + 2040h = 87000080h (DWord) – TCm->VCm

   d. After both above and System Agent DMI TC/VC mapping are programmed, poll VC negotiation pending status until is zero:

      - PCR[DMI] + 201Ah[1]
      - PCR[DMI] + 2026h[1]
      - PCR[DMI] + 2046h[1]

7. Configure OPI PHY (for PCH with OP-DMI ONLY)

   a. PCR[OPIPHY] + B104h[3] = 1b

   b. if (PCR[OPIPHY] + 0110h[20]) == 1b then

      a. PCR[OPIPHY] + B10Ch[6:5] = 11b

   c. PCR[OPIPHY] + 0118h[13:12, 8] = 11b, 1b

   d. PCR[OPIPHY] + 0354h[1] = 1b

   e. PCR[OPIPHY] + 011Ch[14] = 1b

8. Perform various DMI and OP-DMI programming (PM settings, clock-gating …). Most of these settings are required to be programmed before ASPM is enabled.

   a. PCR[DMI] + 2320h[6:4] = 001b
   b. PCR[DMI] + 232C[0][3] =1b (for PCH with OP-DMI ONLY)
   c. PCR[DMI] + 232C[20] = 1b (for PCH with DMI ONLY)
   d. PCR[DMI] + 2320[1] = 1b – Enable DMI PLL shutdown (for PCH With DMI ONLY)

[3] For LP stepping Ax leave this with default value (0b).

OP-DMI specific PM configurations (step e-p for PCH with OP-DMI ONLY)

e. PCR[DMI] + 260Ch[15:0] = 0014h
f. PCR[DMI] + 2614h[8] = 1b
g. PCR[DMI] + 2614h[10] = 1b
h. PCR[DMI] + 2614h[23:16] = 20h
i. PCR[DMI] + 2614h[25] = 1b – done after step 8.g only
j. PCR[DMI] + 2614h[26] = 1b
k. PCR[DMI] + 2614h[27] = 1b
l. PCR[DMI] + 2614h[28] = 1b
m. PCR[DMI] + 2614h[29] = 1b
n. PCR[DMI] + 2614h[30] = 1b
o. PCR[DMI] + 2618h[27] = 1b
p. PCR[DMI] + 2600h[3] = 1b

For PCH with DMI only (step 16-28):

q. PCR[DMI] + 2304h[31:30] = 11b
r. PCR[DMI] + 2304h[18:16] =011b
s. PCR[DMI] + 2478h[31:24] =28h
t. PCR[DMI] + 2344h[31:24] =14h
u. PCR[DMI] + 2344h[23:16] =14h
v. PCR[DMI] + 21A4h[17:15] – If DMI PLL shutdown is enabled (PCR[DMI] + 2320h[1] == 1) then program this field to 4h, else to 2h.
w. PCR[DMI] + 21A4h[14:12] =4h
x. Set Gen1/2/3 Common Clock N_FTS
   i. Set PCR[DMI] + 2340h[7:0] = 2Dh
   ii. Set PCR[DMI] + 2340h[23:16] = 5Bh
   iii. Set PCR[DMI] + 2478h[7:0] = 2Ch
   iv. Retrain DMI link after N_FTS values are set, PCR[DMI] + 21A8h[5] = 1b

y. PCR[DMI] + 2310h[30,17] = 1b,1b (first step, cannot be combined with other steps)
z. PCR[DMI] + 2314h[22:21] = 01b
aa. PCR[DMI] + 2314h[5] = 1b
bb. PCR[DMI] + 2314h[23] = 0b
cc. PCR[DMI] + 2310h[21] = 0b

dd. PCR[DMI] + 2238h[0] = 1b (for PCH With DMI ONLY)
ee. PCR[DMI] + 2238h[11:10] = 10b (for PCH With DMI ONLY)
ff. PCR[DMI] + 2238h[9:8] = 01b (for PCH With DMI ONLY)

gg. PCR[DMI] + 2334h[8] = 1b (for PCH With DMI ONLY)
hh. PCR[DMI] + 2334h[0] = 1b (for PCH With DMI ONLY)
ii. PCR[DMI] + 2334h[1] = 1b
jj. PCR[DMI] + 234Ch[5] = 1b

kk. PCR[DMI] + 234Ch[2:0] = 000b

ll. PCR[DMI] + 2234h[4] = 1b (using byte access)
mm. PCR[DMI] + 2234h[3] = 1b (using byte access)
nn. PCR[DMI] + 2234h[2] = 1b (using byte access)
oo. PCR[DMI] + 2234h[1] = 1b (using byte access)
pp. PCR[DMI] + 2234h[0] = 1b (using byte access)
qq. PCR[DMI] + 223Eh[6] = 1b
rr. PCR[DMI] + 223Eh[5] = 1b
ss. PCR[DMI] + 223Eh[3:2] = 00b

9. Enable Active State PM Control (ASPM). See section 8.3.1 for ASPM requirements on PCH platforms. (for PCH with DMI ONLY)

10. After required DMI programming is performed lock registers
    a. PCR[DMI] + 2618h[25] = 1b (for PCH with OP-DMI ONLY)
    b. PCR[DMI] + 2234h[31] = 1b
    c. Set PCR[DMI] + 2234h[25:24] = 01b together with step b) (for PCH with DMI ONLY)

Lock bits once set cannot be changed until PLTRST# occurs. All DMI Source Decode registers (except I/O Trap) need to be programmed before DMIC.SRL (2234h[31]) is set (refer to section 2.5.1).

§ §

8 PCI Express* Support

From the perspective of PCI Express* hierarchy, the PCH appears as part of a Root Complex with a maximum of six (for Skylake PCH-LP) or sixteen (for Skylake PCH-H)downstream root ports and one integrated endpoint (Intel® High Definition Audio controller). Additionally, it has an internal upstream (North) port that connects to the Processor via the DMI link. This chapter covers PCI Express* System BIOS support specific to PCH. The reader is expected to be familiar with PCI Express* Specification 3.0 and related terminologies (Endpoints, Root port, Isochrony, ASPM, etc.).

The basic assumptions/requirements are:

System BIOS is able to enumerate and configure all root ports as PCI-PCI bridges in compliance with PCI-to-PCI Bridge Architecture Specification, 2.0.

1. System BIOS handles/reports the PCI Express* interrupt routing schemes for the root ports and for the secondary buses behind them correctly (refer to the chapter “PCI Express* Interrupt Routing on PCH for more information).

8.1 PCI Express* Configuration Space Base Address

The PCI Express* Specification defines a block within the memory address space as PCI Express* configuration space addressable through a Bus:Device:Function mapping. The base address of this configuration space is determined by a register in the Processor. Please refer to the relevant Processor EDS and System BIOS Writer’s Guide for the register information.

Once initialized and enabled by System BIOS, software can use memory instructions to access the PCI Express* configuration space registers by byte, word, or DWord as long as the access does not cross DWord boundary.

To maintain the compatibility with PCI Express* configuration space, the first 256 bytes (offset 00h through FFh) of the configuration space for a Bus:Device:Function can also be accessed via the I/O index/data register pair at CF8h/CFCh as defined in PCI Express* 2.x specification.

The memory-mapped physical address of a given PCI Express* configuration register of a specific bus:device:function can be determined by:

- PCI Express* Config Space Base Address + Bus Number * 100000h + Device Number * 8000h + Function Number * 1000h + Register Offset

8.1.1 Register Access Using Capabilities List

PCI Express* configuration registers are defined as a set of capability structures that are linked via Capabilities List pointers. It is recommended that System BIOS follows the pointers of the Capabilities List when accessing configuration registers of individual capability structures, so that the code will be reusable without change in the case where the offset of the capability structures are changed in future chipsets. Here are the guidelines for accessing standard PCI Express* configuration registers:

1. Locate the starting address of the configuration space section for a given Bus:Device:Function .

2. Locate the first Capabilities List pointer at offset 34h, or use offset 100h as the location of the first structure in PCI Express* extended configuration space, and follow the linked pointers until the desired capability structure is found by a matching Capability ID and thus the base address of the capability structure is determined.
3. Add the register offset with the capability structure of the base address found above to read or write the register.

8.1.2 Device/Port Type Field of PCI Express* Devices

The PCI Express* Specification v1.1 states 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, etc.) must be consistent between both ends of a PCI Express* link. It is recommended that software not differentiate PCI Express* Endpoint devices versus legacy PCI Express* Endpoint devices when performing configuration for a link.

Device/Port Type field should be checked/used when there is a need. For example, when software needs to access a register file that only applies to a root port but not to an endpoint device.

8.2 Configuration of PCI Express* Root Ports

There are 12 (Skylake PCH-LP)/ 20 (Skylake PCH-H) PCI Express* downstream root ports on PCH as shown below:

Skylake/Kaby Lake PCH-LP:

Device Number	Function Number	Port Number
28	0-7	1-8
29	0-3	9-12

Note: At most 6 ports that can be enabled at any one time, each of the Port 1-4, Port 5-8 and Port 9-12 to be configured as 4x1, 2x2, 1x2+2x1, 1x4.

Skylake/ Kaby Lake PCH-H

Device Number	Function Number	Port Number
28	0-7	1-8
29	0-7	9-16
27	0-3	17-20

Note: At most 16 ports that can be enabled at any one time.

Note: Whenever the document refers to Dxx:Fn, it refers to the legal combinations above.

For the PCI Express ports, the assignment of a function number to a port is not fixed. BIOS can re-assign the function numbers on a port by port basis. This capability will allow BIOS to disable/hide any root port and have still have functions 0 thru N-1 where N is the total number of enabled root ports.

Port numbers will remain fixed to a physical root port. The existing root port Function Disable registers operate on physical ports (not functions, contrary to what the name implies)

Port Configuration (1x4, 2x2, 4x1, .etc.) is not affected by the logical function number assignment and is associated with physical ports. These root ports can be configured by hardware strapping to operate at different link widths (i.e., different number of lanes per link). Such configurations are reflected/controlled by the Strap and Fuse Configuration (STRPFUSECFG) register at the first root port within a controller, which are RP1, RP5, RP9, RP13 and RP17. See the PCH EDS for the complete register description.

8.2.1 PCIe* Function Disable Requirement

System BIOS must function disable the unavailable ports using the proper bit(s) in the Port Configuration and Disable, PCR[spa~spc] + 00h following the flow below (spa=PCIe Controller 1, spb=PCIe Controller 2, spc=PCIe Controller 3). However if there's a device detected in root port other than function 0, in order to comply with PCI/PCIe* specifications, BIOS can make use of Root Port Function Number (RPxFN) in the same register to reassign Function number associated with the port. If this method is used however, care to be taken on the MPS table interrupt routing table, as the interrupt routing in hardware is tied to the root port, but the table reference it through BDF. If Intel® RST for PCIe* Storage is enabled and the PCIe* storage device remapping is to be done on the root port associated with function 0 (initially), that root port is to be hidden and thus it requires the RPxFN method.

Also system BIOS should make sure at least 15ms delay (after de-assertion of the PLTRST#) before executing PCIe* card detection code.

Note: When disabling a function, only the configuration space is disabled. Software must ensure that all functionality within a controller that is not desired (such as memory spaces, I/O spaces, and DMA engines) is disabled prior to disabling the function. There are registers that is tied to root ports (first root port of PCIe* controller, function 0 or function 4 before any RPxFN remap). If these root ports are disabled/fused off, and programming needs after that needs to be done through the sideband access. The "FID", P2SB PCI offset DAh bit[7:0] needed to access a given RP can be calculated as (DeviceNum <<3 + Port#), where the Port# is a zero-based, in relative to the particular controller instance. To access RP5, D28F4 for instance the FID is (28<<3 + 0).

PSF will have the corresponding Function Disable, Port Config Hide (Not in LP for the Type 1) and Port Function Number registers in its own configuration space for use with fabric decode. BIOS will need to ensure that the PSF and PCIe* RP registers are kept in sync. Refer to section 5.3 for the PSF register details.

For each of the available PCIe* ports:

In early boot, before any PCI scan to be done, refer to section 5.3.1

In later boot stage

If the port PCI config is hidden (cause by the early boot step, because of the Root Port Configuration soft-strap does not map this port to any PHY lanes or the port is function disabled by fused), perform the same steps as in section 8.2.1.1

Else if the port is not mapped to any PHY lanes (Lane Muxing soft-straps and fuses do not map this port to any PHY lanes), function disable the port by, refer to section 8.2.1.1.

Else if the port is not hot plug capable and no PCIe* card is detected, refer to section 8.2.1.1

Else if the port is not hot plug enable and a PCIe* card is detected, and BIOS wants to disable the port, BIOS should set link disable at B0:Dxx:Fn:50h[4] followed by function disable the port, refer to section 8.2.1.1.

Note: If the port is hot plug capable, do not disable the port. If BIOS wants to disable the port, BIOS should not enable the hot plug capability or must disable the hot plug capability of the port.

8.2.1.1 PCIe* Function Disable Handling

Root ports of PCIe* controllers could be disabled for various reasons stated in section 8.2 and section 29.1.

1. Link programming
   a. If card is detected
      i. Set link disable at B0:Dxx:Fn:50h[4]
   b. Else
      i. Set B0:Dxx:Fn:338h[26] = 1b
      ii. Poll B0:Dxx:Fn:328h[31:24] for up to 50ms for value of:
         - 1h for ports that own at least 1 HSIO lane and their 1st lane is configured as PCIE
         - Eh for ports that don't own PCIE lane, either because their lane was configured for another interface such as SATA, or because they don't own any lane at all due to being part of PCIE controller configured as 1x4, 2x2 etc.
      iii. Set B0:Dxx:Fn +408h[27] =1b
2. PCIe* RP IOSF Sideband register offset 00h[19:16], depending on the port that is Function Disabled
3. Then the BIOS need to set the function disable register in PCIe* RP and related PSF register, for example, root port 1:
   a. Set "Port 0 Disable", PCR[SPA] + 2h [0] to 1 (SPA=PCIe Controller 1) using byte access, if only this is not used for PCIe* storage remapping feature.
   b. Set related PSF function disable bit for root port 1, PCR[PSF_1] + "AGNT_T0_SHDW_PCIEN"[RP1][8] to 1
4. If card is detected then disable the respective SRCCLK in PCR[ICC] + 100Ch.

Some power management steps needed along with function disable programming above:

1. PCIe PCI offset E1h[7, 6] to 1, 1 (first root port of the controller)
2. PCIe PCI offset E2h[4] to 1 (first root port of the controller)
3. PCIe PCI offset 420h[31] to 1
4. PCIe PCI offset 424h[8, 7, 5] to 111b

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 in 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.

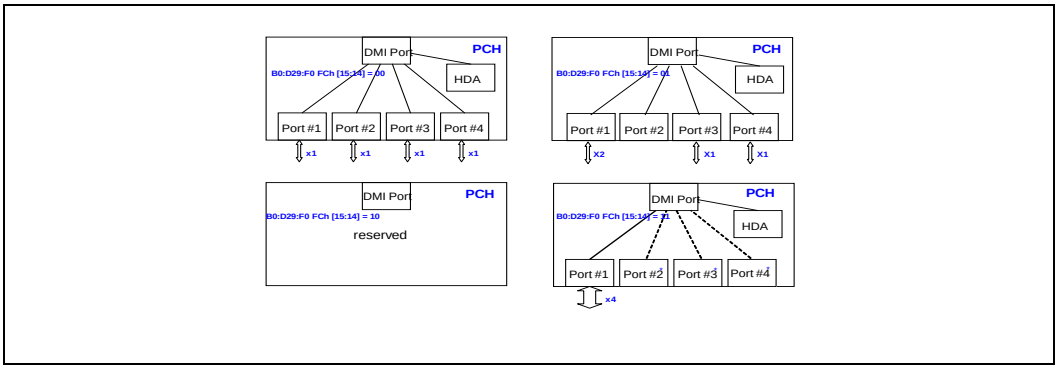
If non ACPI OS is supported, patch up the PIRQ and MP table to reflect the correct interrupt routing information.

8.2.2 Port Width Configuration

Depending on the platform hardware strapping implementation, the PCI Express* downstream root ports of PCH can be configured to different maximum widths and a different number of available ports, as shown in Figure 8-1 below.

The Root Port Configuration Strap (RPC) field (B0:Dxx:F0 + FCh [15:14]) indicates the maximum width the port is configured for and will be translated and reported via the Maximum Link Width (MLW) field of the Link Capabilities register of the PCI Express* Capability Structure of each root port. When RPC = 00b, all 4 ports are available with 1 lane for each port. In all other cases, one or more ports will become unavailable as shown by the dashed lines and shaded boxes in the diagram in Figure 8-1.

Figure 8-1: Platform Controller Hub (PCH) Downstream Root Port Configuration 1



8.2.3 Initialize "Slot Implemented" for Root Ports

The System BIOS must initialize the "Slot Implemented" bit of the PCI Express* Capabilities Register, XCAP Dxx:Fn:42h[8] of each available and enabled downstream root port. Setting this bit will indicate that the PCI Express* link associated with this port is connected to a slot (as compared to being connected to an integrated device component).

8.2.4 Initialize "Physical Slot Number" for Root Ports

The System BIOS must assign a unique number to the Physical Slot Number field of the Slot Capabilities register, SLCAP (Dxx:Fn:54h[31:19] of each available and enabled downstream root ports that implements a slot.

8.2.5 Initialize "Slot Power Limit" for Root Ports

The System BIOS must initialize the Slot Power Limit Value and Slot Power Limit Scale fields of the Slot Capabilities register (Dxx:Fn:54h[16:7]), so cards or modules attached to the root ports will operate properly within the given limit of power consumption. This initialization should take place before the downstream device is enabled.

When either the Slot Power Limit Value or Slot Power Limit Scale field is written, the root port will send a Set_Slot_Power_Limit message to the downstream device, which will capture this information in Captured Slot Power Limit Value and Captured Slot Power Limit Scale fields of its Device Capabilities register and will ensure that it does not exceed the imposed limit of slot power consumption.

The slot power consumption and allocation is platform specific. Please refer to the PCI Express* Card Electromechanical (CEM) Spec for details.

8.2.6 SCI/SMI Generation

To support power management events on non-PCI Express* aware OSes, PM events can be routed to generate SCI. To generate an SCI, set the MPC.PMCE (Dxx:Fn:D8h[31]) bit. When set, a power management event will cause SMSCS.PMCS (Dxx:Fn:DCh[31])to be set.

Additionally, System BIOS workarounds for power management can be supported by setting MPC.PMME, Dxx:Fn:D8h[0]. When set, power management events will set SMSCS.PMMS, Dxx:Fn:D8h[0], and an SMI# will be generated. This bit will be set regardless of whether interrupts or SCIs are enabled. The SMI# may occur concurrently with an interrupt or SCI.

8.2.7 Build Capability Linked List and Extended PCIe* Capability Linked List

1. Build Capability linked list
   a. Read and write back to capability registers 34h, 41h, 81h and 91h using byte access.
   b. Program NSR, A4h[3] = 0b

2. Build Extended PCIe* Capability linked list

   a. NEXT_CAP = 0h

   b. If the RP is GEN3 capable (by fuse and BIOS policy), enable Secondary PCI Express Extended Capability
      i. Set Next Capability Offset, Dxx:Fn +220h[31:20] = NEXT_CAP
      ii. Set Capability Version, Dxx:Fn +220h[19:16] = 1h
      iii. Set Capability ID, Dxx:Fn +220h[15:0] = 0019h.
      iv. NEXT_CAP = 220h

   c. If support L1 Sub-State
      i. Set Next Capability Offset, Dxx:Fn +200h[31:20] = NEXT_CAP
      ii. Set Capability Version, Dxx:Fn +200h[19:16] = 1h
      iii. Set Capability ID, Dxx:Fn +200h[15:0] = 001Eh
      iv. Read and write back to Dxx:Fn +204h
      v. Refer to section 8.3 for other requirements
      vi. NEXT_CAP = 200h

   d. If support ACS
      i. Set Next Capability Offset, Dxx:Fn +140h[31:20] = NEXT_CAP
      ii. Set Capability Version, Dxx:Fn +140h[19:16] = 1h
      iii. Set Capability ID, Dxx:Fn +140h[15:0] = 000Dh
      iv. Read and write back to Dxx:Fn +144h
      v. NEXT_CAP = 140h

   e. If support Advanced Error Reporting
      i. Set Next Capability Offset, Dxx:Fn +100h[31:20] = NEXT_CAP
      ii. Set Capability Version, Dxx:Fn +100h[19:16] = 1h
      iii. Set Capability ID, Dxx:Fn +100h[15:0] = 0001h

ELSE

i. Set Next Capability Offset, Dxx:Fn +100h[31:20] = NEXT_CAP

ii. Set Capability Version, Dxx:Fn +100h[19:16] = 0h

iii. Set Capability ID, Dxx:Fn +100h[15:10] = 0000h

8.2.8 Maximum Payload Size

PCIe* RP supports 128B and 256B maximum payload size.

System BIOS must ensure that downstream posted memory write from processor will never exceed 64B data payload. This can be configured through the downstream posted write chain configuration bit on processor DMI side.

DCTL.MPS is default to 128B MPS on PCIe* RP. System BIOS must enumerate the device connected to the PCIe* RP and configure the DCTL.MPS accordingly. If the device supports 128B MPS only, System BIOS should leave DCTL.MPS at 128B. If the device supports 256B, System BIOS should configure DCTL.MPS to 256B for both the PCIe* RP and device connected to the RP.

1. Program DCTLMPS, RP PCI offset 48h[7:5] to 000b for 128B or 001b for 256B
2. Program IPCS. IMPS, RP PCI offset F0h[10:8] to the same value.

If peer memory writes are supported in the platform and there is at least one device connected to any of the PCH Root Port that supports only 128B max payload size, BIOS must program all the PCH Root Ports such that upstream posted writes are split at 128B boundary by setting CCFG.UPSD to ‘0’ and CCFG.UPRS to “000”. Splitting upstream posted writes at 128B boundary is not a requirement if only peer MCTP messages are supported but not posted memory write.

For hot plug supported ports, regardless of the capability of device attached during boot, leave the DCTL.MPS as 128B

8.2.9 PCIe* Relaxed Order

Enable PCIe* Relaxed Order to always allow downstream completions to pass posted writes. To enable feature set the following registers:

1. Set B0:Dxx:Fn:320h[24:23] = 11b
2. PCR[DMI] + 2314h[31, 7] = [1b, 1b]
3. PSF_1_PSF_PORT_CONFIG_ PG0_PORT0 bit[1] = [1]

8.2.10 Completion Retry Status Replay Enable

Following reset it is possible for a device to terminate the configuration request but indicate that it is temporarily unable to process it, but in the future. The device will return the Configuration Request Retry Status. By setting the “Completion Retry Status Replay Enable”, Dxx:Fn + 320h[22], the RP will re-issue the request on receiving such status.

The BIOS must set this bit before first configuration access to the endpoint.

8.2.11 Peer Memory Access

On platform that does not support peer memory access, perform the following steps

1. Set Dxx:Fn + D0h[26, 25] to [1, 1]

If peer memory writes are supported in the platform, refer to section 8.2.8 on CCFG.UPSD and CCFG.UPRS programming requirement.

8.3 ASPM on DMI and the PCI Express* Root Ports

It is recommended that the System BIOS enable L0s/L1 for both DMI and PCI Express* Root Ports whenever possible. BIOS should set PCR[DMI] + 2304h[10] to 0b prior to enabling DMI ASPM and set PCR[DMI] + 2304h[10] back to 1b after enabling DMI ASPM.

Note: Some PCIe* Option ROM may be unstable with ASPM enable. Alternative is to configure ASPM after Option ROM executed.

Note: We recommend PCH platforms to enable L0s and L1, but unless both sides of the link have L0s and/or L1 enabled they will be disabled by the link. L0s must be disabled for Gen3 link, this is applied to DMI/PCIe. System BIOS Software Flow for Enabling ASPM

Before determining whether ASPM can be enabled or not, the System BIOS must perform the following steps:

1. Update the Link Capabilities of the DMI link to indicate L0s/L1 is supported by programming the LCAP Register, PCR[DMI] + 21A4h[11:10] = 11b. (for PCH with DMI ONLY)
2. Enable L1 and disable L0s on DMI by setting PCR[DMI] + 21A8h[1:0] to 10b. (for PCH with DMI ONLY)
3. For each root port, read the Slot Clock Configuration bit, Dxx:Fn:52h[12], of the root port and the endpoint device connected to the port (i.e., D0:F0 on the secondary bus behind the root port). If both components have this bit set, then the System BIOS should set the Common Clock Configuration (CCC) bit, Dxx:Fn:50h[6], for both components at both sides of the link to indicate that components at both ends of the link use a common clock source.
4. If the CCC bit was changed by the System BIOS in step 3, System BIOS should initiate a link training by setting the Retrain Link (RL) bit, Dxx:Fn:50h[5], and then poll the Link Training (LT) bit, Dxx:Fn:52h[11], until it is clear. Note that System BIOS should save and restore CCC bit on S3.

When enabling L0s / L1 support, BIOS should enable upstream device before downstream device. When disabling ASPM, BIOS should make sure downstream device is disabled before upstream device. The System BIOS must perform the following steps to enable L0s/L1 on the root ports:

1. Determine whether the endpoint supports L1 by checking the Active State Link PM Support field of the endpoint Link Capability Register. If the endpoint does not support L1, the System BIOS can skip the L1 calculations below. Likewise, System BIOS should not enable L1 on the root port or the endpoint if the endpoint does not support L1.
2. Calculate the total L0s and L1 exit latency. A description of this calculation is provided in Section 8.3.1.

3. Compare the calculated total exit latency with Endpoint L0s/L1 Acceptable Latency read from the Device Capabilities Register of the Endpoint to determine if L0s or L1 can be enabled for all or some of the links on the entire path to satisfy the Acceptable Latency reported by the Endpoint. The Exit Latency fields reported by the registers are given as a range. It is recommended that System BIOS uses the high end of the range for the latency calculation and comparison. For example, if the latency field reports "2 us to less than 4 us", then 4 us should be used for the calculation.
4. If the comparison in step 3 indicates L0s and L1 can be enabled on a root port and the endpoints attached to the root port, set the root port register Dxx:Fn:E8h[1], set the APMC field, Dxx:Fx:50h[1:0] to 11b and write the same value to the APMC field of the endpoint Link Control register. If the comparison in step 1 indicates only L0s can be enabled on a root port and the endpoints attached to the root port, then set the APMC field, Dxx:Fn:50h[1:0] to 01b and write the same value to the APMC field of the endpoint Link Control register.
5. Prior to enabling ASPM L1, set Dxx:Fn:E2h[4] = 1b. If per-port CLKREQ# is supported, then disable L1.LOW by setting Dxx:Fn:420h[17] = 0b if L1.SNOOZ and L1.OFF (L1 Sub-States) are supported , set Dxx:F0:420[30, 29] = [1b, 1b], and If dedicated CLKREQ# per-port is supported on all enabled ports, set Dxx:F0:E1h[6] = 1b prior to enabling ASPM L1. Note: for each PCIe* controller, only its first root port register field E2h[4] and E1h[6] value are used, thus program them using sideband access
6. If both root and endpoint's L1 Sub-States Extended Capability Offset + 04h[4:0] are 1111b, then:
   a. Read L1 Sub-States Extended Capability Offset + 04h[3:0], and set the smallest common subset supported by both PCIe* rootport and device to L1 Sub-States Extended Capability Offset + 08h[3:0]
   b. Read L1 Sub-States Extended Capability Offset + 04h[15:8], and set the highest value advertised between PCIe* rootport and device to L1 Sub-States Extended Capability Offset + 08h[15:8] on the device
   c. Read L1 Sub-States Extended Capability Offset + 04h[23:19] and [17:16], and set the highest value advertised between PCIe* rot port and device to L1 Sub-States Extended Capability offset + 0Ch[7:0] on both PCIe* root port and device
   d. Set L1 Sub-States Extended Capability Offset + 08h[31:29] to 010b for both PCIe* root port and device
   e. Set L1 Sub-States Extended Capability Offset + 08h[25:16] to A0h for both PCIe* root port and device
   f. Set Dxx:Fn:420h[0] to 1b prior to L1 enabling
   g. Set PCR[ICC] +1000h[23:0] to a value (in time unit) bigger than Tpoweron from step c. For the hot plug port but empty, set it to 45us just in case SMI is not enabled to handle this.
7. If L1.SNOOZ and L1.OFF (L1 Sub-States) are not supported, and per-port CLKREQ# is supported, and LTR is supported:
   a. Set Dxx:Fn:420[13:4] = 32h
   b. Enable L1.LOW by setting Dxx:Fn:420[17] = 1b
   c. Set Dxx:Fn:420h[0] to 1b

8. If the PCIe root port CLKREQ# does not supported and does not assigned, PCIe L1.LOW must be disabled by setting Dxx:Fn:420[17] = 0b.
9. BIOS shall not program PCIe L1 PM Substates Common_Mode_Restore_Time in upstream ports.

8.3.1 Calculation of L-State Exit Latency

Prior to enabling L-state entry, the System BIOS must calculate the total exit latency along the entire path to make sure that the total exit latency is within the tolerance by the endpoint device in order to ensure the functionality and performance of the device.

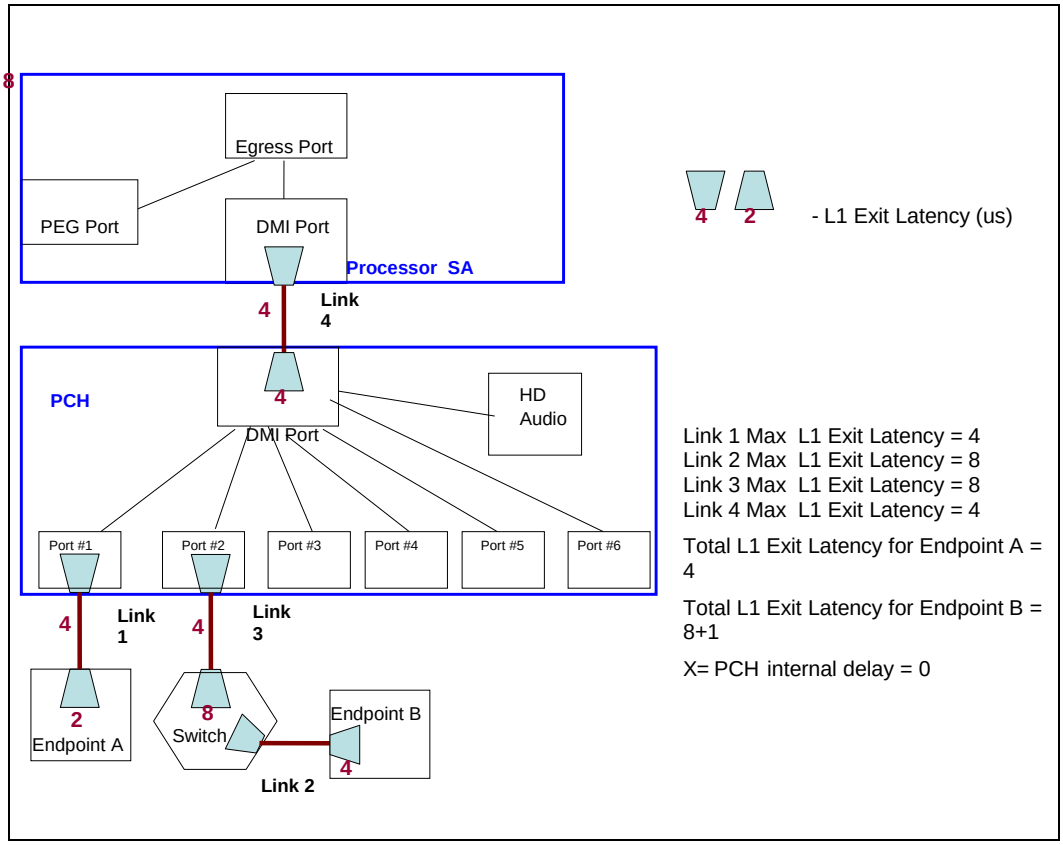
The same method for determining the exit latency can be used for L0s and L1. Figure 8-2 shows an example L1 calculation for clarification purposes.

The exit latency for a link is the longer latency of the two ports at both ends of the link. In the example shown in Figure 8-2, the L1 exit latency for Link1 in the example is 4 although Endpoint E has a shorter exit latency of 2.

When a switch exists along the path, the switch must initiate L1 exit on its upstream port within 1 microsecond after it receives a L1 exit request at any of its downstream ports. The switch must initiate an L0s exit on its upstream port as soon as (without delay) it detects an exit from L0s at any of its downstream ports. Therefore, the internal propagation delay is 0 for L0s exit, as opposed to 1 us in the case of L1 exit. Refer to the PCI Express* specification for more details.

To calculate the total latency along the path, simply add together the Exit Latencies of each link along the path. Given the above, the total L1 exit latency for Endpoint A and B are calculated and shown in Figure 8-2.

Figure 8-2: L1 Total Exit Latency Calculation Example



8.4 SERR# Generation

SERR# may be generated via two paths – through PCI Express* mechanisms involving bits in the PCI header, or through PCI Express* mechanisms involving bits in the AER-like structure. These errors are put together to signal SERR# to the platform as shown in Figure 8-3. As can be seen, the PCI Express* methods for calculating SERR# do not set PSTS.SSE.

For the PCI Express*-based SERR#'s (i.e., the lower half of the diagram), the corresponding error enable bits in the AER-like structure registers for Correctable Error, Uncorrectable-Fatal Error and Uncorrectable-Non-Fatal Error are all disabled by default after reset, so such errors will not be able to generate SERR#. To enable SERR# generation by such errors at root ports, software must program the following registers to enable individual errors:

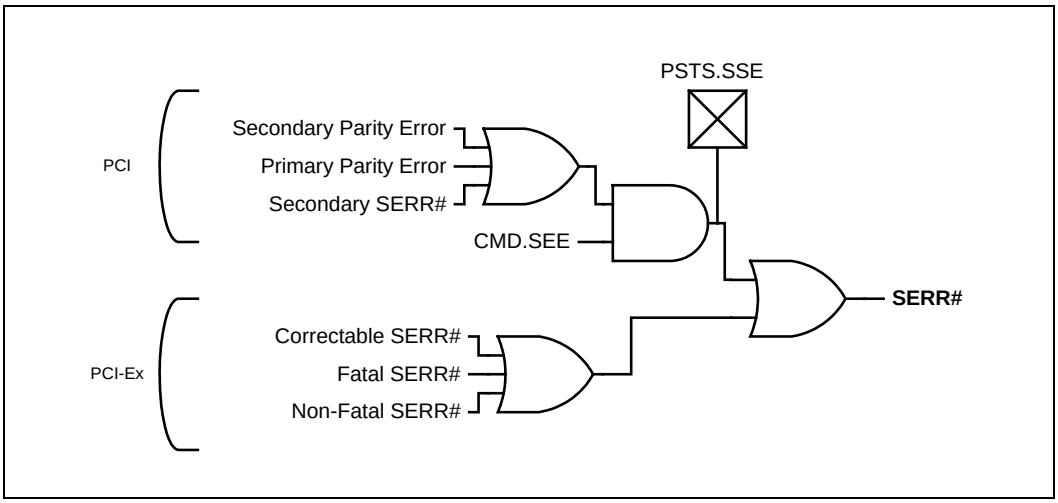
- Device Control register of the root port PCI Express* Capability structure (Dxx:Fn:48h[2:0]).
- Root Control register of the root port PCI Express* Capability structure (Dxx:Fn:5Ch[2:0]).

If the System BIOS is required to enable PCI Express*-based SERR# generation, care must be taken when enabling such errors so that no unwanted errors cause SERR#. In addition to the Device Control and Root Control registers mentioned above,

- The Uncorrectable Error Severity register, Dxx:Fn:14Ch, can be used to map individual errors to Fatal or Non-Fatal groups.
- Correctable Error Mask, Dxx:Fn:154h, and Uncorrectable Error Mask, Dxx:Fn:148h, registers can be used to mask (block) individual error reporting.

Refer to PCI Express* Specification for multi-level control of PCI Express* error signaling.

Figure 8-3: Generation of SERR# to Platform



8.5 PCI Firmware Spec 3.0 Support

PCI Firmware Specification v3.0 expanded the previous version of the spec to include support for the PCI Express* enhanced configuration mechanism, among other things. System BIOS must be compliant with this spec in order to support PCI Express* aware operating systems and PCI Express* aware option ROMs.

8.6 ACPI Table and Methods for PCI Express* Support

Refer to the PCI Firmware Specification for details.

8.6.1 _OSC() Method

Although optional per PCI Firmware Spec v3.0, the _OSC() ACPI control method is required for PCH platforms. Refer to the PCI Firmware Specification for more details…

8.7 PCI Express* PME BIOS Support

PCI Express* delivers PME events by way of in-band Transaction Layer PME messages as opposed to the side-band signaling approach used by PCI Express* bus.

Additionally, the Root Complex can signal a PME event via an interrupt, allowing software to handle the PME as an interrupt event.

8.7.1 Native PME Software Model

PCI Express* aware software can enable a mode where the Root Complex signals PME via an OS native interrupt. When configured for native PME support (which requires that legacy GPE-based PME support be disabled), a Root Port receives the PME Message and sets the PME Status bit in its Root Status register, Dxx:Fn:60h[16]. If software has set the PME Interrupt Enable bit in the Root Control register, Dxx:Fn:5Ch[3] the Root Port then generates an interrupt. The software handler for this interrupt can determine which device sent the PME Message by reading the PME Requester ID field in the Root Status register, Dxx:Fn:60h[15:0]. The interrupt is cleared by setting the PME Status bit, Dxx:Fn:60h[16].

The native PME software model is expected to be used by future PCI Express* aware OS-level software, therefore its implementation details are beyond the scope of this document.

Note: Although it is a PCI Express* Root Complex Integrated endpoint, the Intel® High Definition Audio controller in PCH is not capable of supporting the native PME software model. Its PME is supported the same as a PCI PME. Refer to section 9.3 for details.

8.7.2 Legacy PME Software Model

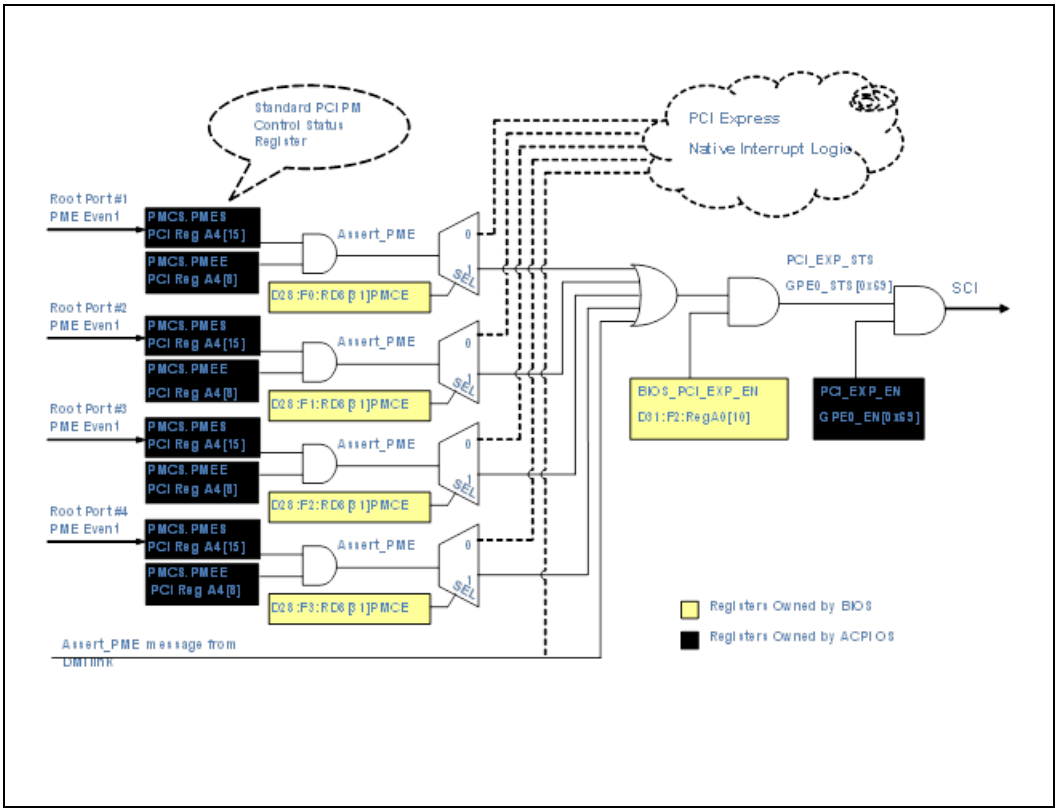
Legacy software\OSs which are unaware of PCI Express* will not understand this in-band message-based, interrupt-driven mechanism for signaling PME. In the presence of legacy OS software, the system power management logic in the Root Complex receives the PME Message and informs system software through an implementation specific mechanism.

PCH has included logic that allows software implementation of PCI Express* PME support using an existing GPE-based ACPI model in a legacy OS environment.

8.7.3 BIOS Enabling of Platform Controller Hub (PCH) PCI Express* PME SCI Generation

PCH provides an option for software to route PCI Express* PME events to the ACPI General Purpose Event (GPE0) register for the purpose of generating an SCI. The diagram below in Figure 8-4 shows how PCI Express* PME events are propagated to an SCI, and the enable controls along the path. As shown in the diagram, some of the registers are owned and programmed by System BIOS, while others are owned by the ACPI OS.

Figure 8-4: Platform Controller Hub (PCH) PCI Express* PME to SCI Path



System BIOS enables SCI generation of PCI Express* PME messages on a PCH root port by:

- Make sure that PME Interrupt Enable bit, Dxx:Fn:5Ch[3] is cleared.
- Program Misc Port Config (MPC) register at PCI Express* config space offset D8h as follows:
  - Set Power Management SCI Enable bit, Dxx:Fn:D8h[31]
  - Clear Power Management SMI Enable bit, Dxx:Fn:D8h[0]

Note: For normal PME operation of an ACPI OS, PME SMI should be disabled.

- Make sure GPE0_STS (ABASE + 8Ch[9]), PCI_EXP_STS is 0 (clear it if not zero).
- Set BIOS_PCI_EXP_EN bit, PMC PCI offset A0[10], to globally enable the setting of the PCI_EXP_STS bit by a PCI Express* PME event.

Note: PCI Express* devices on the Processor SA may also support PME, and certain chipset registers may need to be programmed to enable it. Refer to the proper Processor SA documentation for details.

8.7.4 Handling PCI Express* PME SCI Event

PCI Express* PME events can be handled using the existing GPE model defined by the ACPI spec under an ACPI OS environment, where System BIOS plays an important role.

8.7.4.1 General Mechanism and Sequence

The PCI_EXP_STS/PCI_EXP_EN bit-pair in the GPE0 register maps PCI Express* PME to an ACPI general purpose event which is signaled by an SCI. ACPI OS and System BIOS can handle this event in a co-operative manner as shown in the following sequence:

Note: This is an example to illustrate the main control flow. Although Implementations on different systems may vary and may involve more ACPI objects/methods, the basic control flow described here will still apply.

- System BIOS enables SCI generation of PCI Express* PME messages on a PCH root port. See previous sub section for details.
- System BIOS exposes _PRW object under the root port devices, indicating the wake capability of the device and the bit location of PCI_EXP_STS event in the GPE0 register. (A conceptual ACPI name space tree is shown in the example ASL code at the end of this chapter.)
- System BIOS provides a _GPE._L69 () control method in ASL code as the handler for this event.
- Based on its own power management policy and at a time of its own choosing, the ACPI OS enables the PCI_EXP_EN bit, ABASE + 9Ch[9]. It also sets the PMCS PMEE bit, Dxx:Fn:A4[8], of each root port to enable PME generation.
- When a PCI Express* PME event occurs at any one of the root ports or is received from downstream devices, the PCI_EXP_STS bit, ABASE + 8Ch[9], is set, causing an SCI to be generated.
- The ACPI OS clears the PCI_EXP_EN bit, ABASE + 9Ch[9], then calls the _GPE._L69() control method provided by BIOS to handle this event.
- The ACPI OS clears the PCI_EXP_STS bit, ABASE + 8Ch[9], and sets the PCI_EXP_EN bit, ABASE + 9Ch[9].

8.7.4.2 System BIOS GPE Handler for PME Event

System BIOS needs to provide the _GPE._L69() 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 PCH root port:

1. Read the PME Status bit (Dxx:Fn:60h[16]) of the root port. If the bit is clear, then go to step 5.
2. Clear the PME Status bit, Dxx:Fn:60h[16], by setting it.
3. 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 step 2.

4. Clear the PMCS status bit (Dxx:Fn:DCh[31]) by setting it.

5. Perform a "Notify(Device, 02)" to notify the OS of the event that occurred at this particular root port location.

6. Go to the next root port and repeat steps 1 through 5, until all root ports are done.

Note: Typically the _GPE._L69() method will also handle the PME Events received from the DMI link, which means that the above process would include all PCI Express* ports on the Processor as well.

8.7.5 Transition from Legacy to Native PME Software Model

As previously mentioned, a PCI Express* aware OS will be able to handle PCI Express* PME events using the native, interrupt-driven PME software model. To enable such OS behavior, System BIOS must provide a proper _OSC control method in the ACPI name space for the transition from legacy to native PME software model.

The basic scenario of using _OSC() for native PME support transition is:

- System BIOS provides _OSC() under a proper device in the ACPI name space. For the purpose of native PCI Express* PME support transition, the _OSC can be placed under the system bus scope, i.e., _SB._OSC().
- The OS detects the presence of this _OSC() object during ACPI initialization, and calls the _OSC() with its current capabilities.
- In the _OSC() method, System BIOS ASL code does the following:

{

Examine the input argument "OS capabilities";

If ("OS supports PCI Express Native PM" flag ==1)

{

- Clear the PMCE bit, Dxx:Fn:D8h[31], for all the root ports to route PMEs to native interrupt logic.
- Disable generation of SCI by PME event on the Processor SA. Please refer to Processor SA BIOS Writer's Guide for details.
- Clear the BIOS_PCI_EXP_EN bit, PMC PCI offset A0[10], to globally disable PCI Express PME events.

}

Return the Capabilities with "OS Supports PCI Express Native PME" flag unchanged, indicating that System BIOS is handing over PCI Express* PME support to exclusive OS control.*

}

OS examines the value returned by _OSC() method and sees that “OS Supports PCI Express* Native PM” flag stays set; therefore it enables the native PME software model.

8.7.6 WAKE# Support

PCH supports the PCI Express* WAKE# side-band signal pin. When the WAKE# pin is activated and detected at one or more root ports while the system is in a sleep state, an internal signal is sent to the power management controller to cause the system to wake up.

Note: This internal signal does not cause an interrupt or GPE to be generated.

When the system has returned to a working state from a previous low power state, the device requesting service will send a PM_PME message continuously, until acknowledged by the root port.

8.8 PCI Express* Hot Plug BIOS Support

The PCI Express* architecture is designed to natively support both hot plug and hot-removal of devices based on a standard usage model with well-defined behavior and interface for indicators, push buttons, and MRLs. PCH supports module based Hot-Plug (i.e., ExpressCard*). Hot-plug is detected by the Presence Detect pin, and hot-removal is usually initiated by the Attention Button (if implemented) or via a GUI applet at OS level. Note that “surprise-removal”, although supported at hardware level to protect the electrical components, may not be supported by the system software for obvious reasons.

Assuming a root port implements a slot with hot plug capability, the following registers should be used as the hot plug interface to software (see the PCH EDS for a full description of the registers):

- Slot Capabilities Register, Dxx:Fn:54h – Reports port/slot capabilities to software.
- Slot Control Register, Dxx:Fn:58h – Control indicators and enable interrupts.
- Slot Status Register, Dxx:Fn:5Ah – Report slot status and hot plug events.

BIOS should also make sure the L0s is disabled on empty slots prior booting to OS. Optionally, BIOS can also intercept Presence Detect Changed interrupt to enable L0s on hot plug with the L0s supported devices and disable L0s on hot unplug on those device.

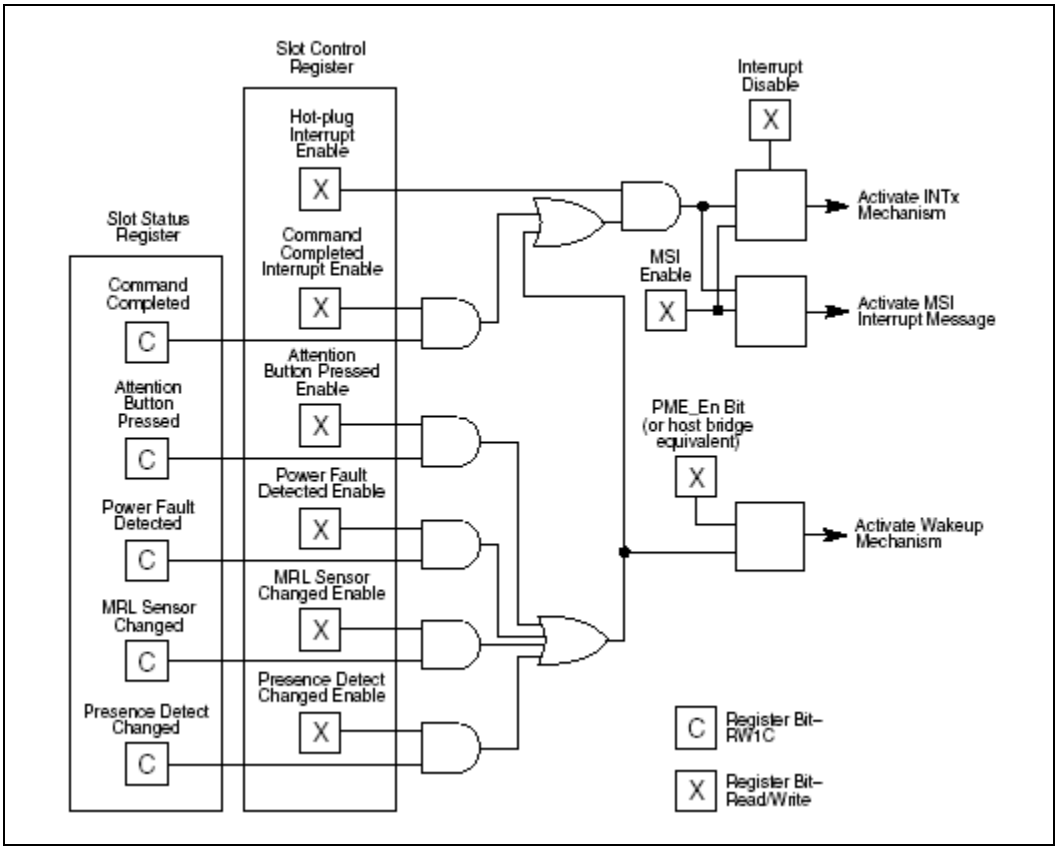
8.8.1 Native Hot Plug Software Model

PCI Express* aware driver/OS can enable a mode where a hot plug event is signaled by an interrupt. When configured for native hot plug support, a root port detects a hot plug event at its interface and sets the proper bit in its Slot Status register. If enabled via the corresponding bit in Slot Control register, the root port will generate an interrupt to signal the hot plug event. The software servicing the interrupt can determine the cause of the interrupt by examination of the Slot Status register. After handling the event, it dismisses the interrupt by writing a 1 to the bit in the Slot Status register.

The figure below illustrates the interrupt-driven native hot plug software model.

The native hot plug software model is expected to be used by OS-level software, therefore its implementation details are beyond the scope of this document.

Figure 8-5: Native Hot Plug Software Model



8.8.2 Legacy Hot Plug Software Model

Legacy software\OS which is unaware of PCI Express* will not understand the PCI Express* native hot plug software interface model. In the presence of legacy OS system software, the root port signals a hot plug event to the Root Complex which informs system software of the event through an implementation specific mechanism.

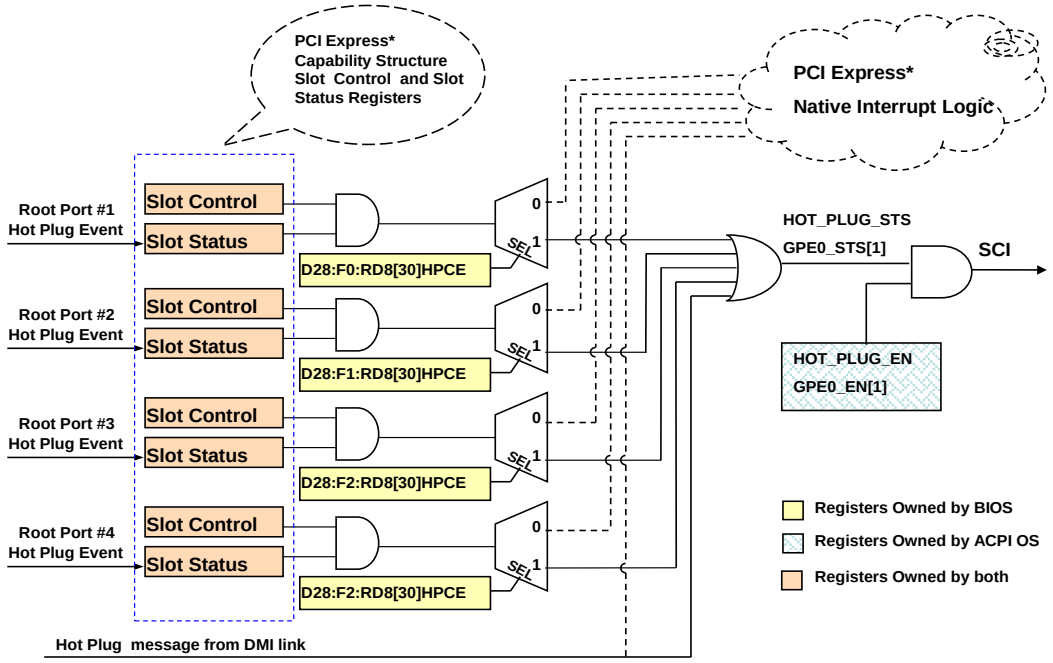
PCH has included logic that allows software implementation of PCI Express* hot plug support using the existing GPE-based ACPI model in a legacy OS environment.

8.8.2.1 System BIOS Enabling of Platform Controller Hub - PCI Express* Hot Plug SCI Generation

PCH provides an option for software to route PCI Express* hot plug events to the ACPI General Purpose Event (GPE0) register to generate an SCI. The diagram below shows how PCI Express* hot plug events generate an SCI, and the enable controls along the path. As shown in the diagram of Figure 8-6, some of the registers are owned and

programmed by System BIOS, some are owned by ACPI OS, and some are owned by both.

Figure 8-6: Platform Controller Hub - PCI Express* Hot Plug to SCI Path



- PCH PCI Express Hot-Plug to SCI Path -

System BIOS enables SCI generation by PCI Express* hot plug events on a PCH root port:

Note: If Hot Plug is supported, then write a 1 to the Hot Plug Capable (bit6) and Hot Plug Surprise (bit5) in the Slot Capabilities register, Dxx:Fn:54h. Otherwise, write 0 to the bits.

- Clear the following status bits, by writing a 1 to them, in the Slot Status register, Dxx:Fn:5Ah:
  - Presence Detect Changed (bit3)
- Program the following bits in Slot Control register, Dxx:Fn:58h:
  - Presence Detect Changed Enable (bit3) = 1b
  - Hot Plug Interrupt Enable (bit5) = 0b

Note: Power Fault Detected Enable (bit1) and MRL Sensor Changed Enable (bit2) are not supported on PCH, so these bits have no effect.

- Program the Misc Port Config (MPC) register, Dxx:Fn:D8h as follows:
  - Hot Plug SCI Enable (HPCE, bit30) = 1b
  - Hot Plug SMI Enable (HPME, bit1) = 0b

Note: For normal hot plug operation under an ACPI OS, hot plug SMI should be disabled.

- Clear the GPE0_STS, (ABASE + 8Ch[1]), HOT_PLUG_STS bit.

Note: PCI Express* devices on the Processor SA may also support hot plug, and certain chipset registers may need to be programmed to enable it. Refer to the proper Processor SA documentation for details.

8.8.2.2 Handling PCI Express* Hot Plug SCI Event

PCI Express* hot plug events can be handled using the existing GPE model defined by ACPI specification under ACPI OS environment, where System BIOS plays an important role.

8.8.2.2.1 General Mechanism and Sequence

The HOT_PLUG_STS/HOT_PLUG_EN bit-pair in GPE0 maps PCI Express* PME to an ACPI general purpose event which is signaled by an SCI. ACPI OS and System BIOS can handle this event in a co-operative manner as shown in the following sequence. Note that this is an example to illustrate the main control flow. Although implementations on different systems (especially for mobile systems with docking support) may involve more ACPI objects/methods, the basic control flow described here will still apply.

- System BIOS enables SCI generation of PCI Express* hot plug events on a PCH root port. See previous sub section for details.
- System BIOS exposes a child Device object under the root port Device object, and exposes _EJx() and _STA() control methods under the child device scope, indicating the hot plug capability of the child device. (A conceptual ACPI name space tree is shown in the example for ASL code at the end of this chapter.)
- System BIOS exposes a _GPE._L61 () control method in ASL code as the handler for this SCI event. This causes the ACPI OS to enable the HOT_PLUG_EN bit in the GPE0_EN, ABASE + 9Ch[1].
- When a PCI Express* hot plug event occurs at the hot plug-enabled root port, HOT_PLUG_STS bit in GPE0_STS, ABASE + 8Ch[1], is set, generating an SCI.
- The ACPI OS clears the HOT_PLUG_EN bit, ABASE + 9Ch[1], then calls the _GPE._L61() control method provided by BIOS to handle this event.
- System BIOS _GPE._L61() method determines the cause of the SCI, performs proper actions accordingly, and then notifies the ACPI OS of the proper cause of the event at a proper device location.
- The ACPI OS clears the HOT_PLUG_STS bit, ABASE + 8Ch[1], and re-enables the HOT_PLUG_EN bit, ABASE + 9Ch[1].
- Based on the event notification received from System BIOS, the ACPI OS will take actions accordingly, which may involve calling System BIOS _EJx(),

_STA() methods for device removal, or enumerating the device which has just appeared.

8.8.2.2.2 System BIOS GPE Handler for Hot Plug Event

System BIOS needs to provide the _GPE._L61() control method in the ACPI name space as the actual handler for PCI Express* hot plug SCI event. Once invoked by the ACPI OS, this control method should perform the following:

For each PCH root port that is hot plug-enabled,

1. Read the HPCS bit of SMSCS Register at Dxx:Fn:DCh[30]. If HPCS Status bit=0, then the port has no hot plug event detected, go to step 6. Otherwise set this bit to clear it.
2. Examine the Slot Status register at Dxx:Fn:5Ah to determine the cause of the hot plug event at this port, according to the table below:

Attention Button Pressed (ABP,Bit0)	Presence Detect Changed (PDC,Bit3)	Presence Detect State (PDS,Bit6)	Hot Plug Event Type	BIOS Action
x	1	1	Device Inserted	Notify OS of insertion event (Device Check 01h)
x	1	0	Surprise Removal Occurred	Notify OS of removal event (Device Check 01h) [1]
1	0	1	Device Removal Request	Notify OS of removal request (Eject Request 03h)
1	0	0	Device Insertion Request	Implementation-specific [2]
0	0	x	No event	Do nothing (Ignore) [3]

Notes:
1. System failure may occur because OS may not support such surprise removal.
2. One example is to blink Attention Indicator to signal system readiness for device insertion.
3. These are abnormal conditions by either user or system error.

3. Clear the ABP and PDC bits in the Slot Status register, Dxx:Fn:5Ah[3,0] by setting them.
4. Depending on the hot plug event type, perform a “Notify (Device, 01)” or “Notify (Device, 03)” to notify OS of the event or ignore the event by doing nothing.
5. Depending on the hot plug event type and the availability of the indicators reported by the Slot Capabilities register, Dxx:Fn:54h of the root port and the Device Capabilities register of the Endpoint device, program the Power Indicator Control field, Dxx:Fn:58h[9:8] and/or the Attention Indicator Control field, Dxx:Fn:58h[7:6] to activate/de-activate these indicators.
6. Go to the next port and repeat steps 1-5, until all ports are done.

Note: Typically the _GPE._L61() method will also handle the hot plug events received from the DMI link, which means that the above process would include PCI Express* ports that are hot plug-enabled on the Processor SA as well.

8.8.2.2.3 System BIOS _EJx() and _STA() Control Methods

System BIOS needs to provide _EJx() and _STA() control methods for removable devices, as defined in the ACPI Spec. The _EJx() method is called by OS when it receives an "Eject request" notification from System BIOS and usually performs device-specific tasks to prepare the device for removal. The _STA() method is called by the OS during the process of insertion or removal handling, and is expected to return the accurate current status information of the device. Please refer to the latest ACPI Specification for more information.

8.8.3 Transition from Legacy to Native Hot Plug Software Model

As previously mentioned, a PCI Express* aware OS will be able to handle PCI Express* hot plug events using the native, interrupt-driven hot plug software model. To enable such OS behavior, System BIOS must provide a proper _OSC control method in the ACPI name space for the transition from legacy to native hot plug software model.

The basic scenario of using _OSC() for native hot plug support transition is:

- System BIOS provides _OSC() under the root port device scope in the ACPI name space. For example, _SB.PCI0.EXP1_OSC().
- The OS detects the presence of this _OSC() object during ACPI initialization, and calls the _OSC() with its current capabilities.
- In the _OSC() method, System BIOS ASL code does the following:

  For all PCH root ports that have been enabled for hot plug support:

  {

  Examine the input argument "OS capabilities";

  If ("OS supports PCI Express Native Hot Plug" flag ==1)*

  {

  - Clear the HPCE bit, Dxx:Fn:D8h[30], to route hot plug events to native interrupt logic.
  - Clear the following status bits, by writing 1b to them, in the Slot Status register, Dxx:Fn:5Ah:
    - Attention Button Pressed (bit0)
    - Presence Detect Changed (bit3)

  }

  Return the Capabilities with "OS Supports PCI Express Native Hot Plug" flag unchanged, indicating that System BIOS is handing over PCI Express* hot plug support to exclusive OS control.*

  }

- OS examines the value returned by _OSC() method and sees that "OS Supports PCI Express* Native hot plug" flag stays set, therefore it enables the native hot plug software model.

Note: For the PCI Express* hot pluggable root ports on the Processor SA _OSC methods are also needed under the scope of such root ports. Refer to the proper Processor SA BIOS Writer's Guide for details.

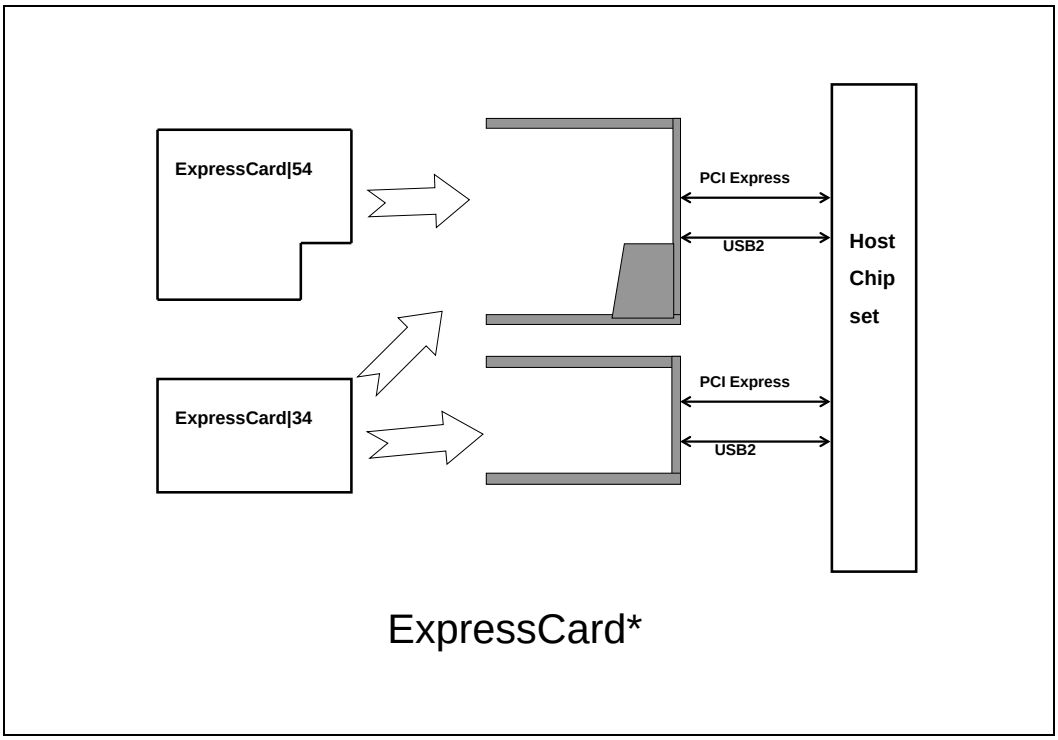
8.8.4 PCIe* Device Hot Plug Enabling

1. Enable PCIe* hot plug SMI for hot plug enabled port(s) by setting Dxx:Fn:D8h[1]
2. In SMI Handler:
   a. Check if Present Detect State Dxx:Fn:5Ah[6] = 1
   b. Enable Common Clock Configuration and ASPM according to the flow in Section 8.3.1

8.9 ExpressCard* Support

ExpressCard* is an interface specification for small, modular add-in cards. An ExpressCard* slot connects to both the PCI Express* and USB 2.0 interfaces, as shown in Figure 8-7. Refer to ExpressCard* specification for details.

Figure 8-7: ExpressCard* Slot Conceptual Diagram



In order to support ExpressCard* implementations, usually on mobile platforms, utilizing both the PCI Express* interface and the USB 2.0 interface at the same time, the operating system must be able to determine the dependencies between the two interfaces on an ExpressCard* slot. Without understanding the inter-dependencies of the two interfaces, executing an eject request to turn off one device may cause undesirable effects on the other device. The mechanism used to expose the

dependency is an ACPI object, _EJD, and is provided by System BIOS based on the design of the specific host platform.

ASL sample code is given below to help illustrate the usage of _EJD objects for ExpressCard* support.

```
Scope(\_SB) {
    Device(PCI0) {                                  // Root PCI Bus
    Name(_HID, EISAID("PNP0A03"))                  // _HID for root device
    Name(_ADR,0)                                   // Device 0 on this bus

    Device (P2P1) {// PCI-to-PCI bridge (may or may not be Express device)
          Name(_ADR,0x000C0000) // Device#Ch, Func#0 on bus PCI0

                Device(USB1) {                                    // USB Host
Controller
                Name(_ADR,0x00000000) // Device#0h, Func#0 on bus P2P1

                       Device(RHUB) {            // USB root hub
                       Name(_ADR,0x00000000)

                           Device(PRT0) { // USB port linked to the ExpressCard*
slot.
                           Name(_ADR,0x00000000)
                           Name(_EJD,"\_SB.PCI0.XCRD.S1F0")  // Eject Dependency
                           }

                           Device(PRT1) {          // Another USB port
                           Name(_ADR,0x00000001)
                           }
                       } // end RHUB
                  } // end USB1
             } // end P2P1

            Device (XCRD) { // PCI Express Port that connects to ExpressCard*slot)
            Name(_ADR,0x000E0000)          // Device#Eh, Func#0 on bus PCI0
            Name(_HPP, Package()
                        {0x08,0x40, 0x01, 0x00})  // Hot Plug Parameters package

                // Device definition for Slot1- MODULE SLOT
                Device (S1F0) {               //  Slot1, Func#0 on bus NCRD
                Name(_ADR,0x00000000)
                Method(_EJ0, 1) {                 // eject method }
                Name(_EJD,"\_SB.PCI0.P2P1.USB1.RHUB.PRT0") // Eject dependency
                }

                Device (S1F1) {               // Slot 1, Func#1 on bus NCRD
                Name(_ADR,0x00000001)
                Method(_EJ0, 1) {                 // eject method }
                }

                ...             // other functions of the slot

                // note that to match up the PCI Express side and the USB side via
                // _EJD, only function 0 is required. Other functions may be
                // required for other reasons (such as _EJ0 as shown).

             } // end XCRD

  } // end PCI0
} // end Scope (\_SB)
```

Note: When surprise removal or hot plug is supported on a PCI Express* link, Completion Timeout errors may occur and it is necessary for system software to handle this scenario. This is not something that is unique to PCH, but applies to any PCI Express* link that supports surprise removal.

There are two options to avoid NMI resulting from SERR# generated by the Non-Fatal Completion Timeout (CT) error; they are described below:

1. Mask Completion Timeouts as a source of a Non-Fatal error that can generate SERR.

BIOS should mask the reporting of Completion timeout (CT) errors by setting the uncorrectable Error Mask register Dxx:Fn:108h[14].

2. Mask All Non-Fatal Errors as a source of generating SERR, of which Completion Timeout is one possible source. Use the PCI Express* Control registers which have separate enables for Fatal, Non-Fatal, and Correctable errors instead of using the PCI SERR# enable which allows SERR# for both Fatal and Non-Fatal errors. BIOS should:
   - Clear the SERR# Enable bit, Dxx:Fn:04h[8].
   - Set the Fatal Error Reporting bit, Dxx:Fn:48h[2].
   - Clear the Correctable Error, Non-Fatal Error, Unsupported Request Reporting Enable bits, Dxx:Fn:48h[0, 1, 3].
   - Set the System Error on Fatal Error bit, Dxx:Fn:5Ch[2].
   - Clear the System Error on Correctable Error and Non-Fatal Error Enable bits, Dxx:Fn:5Ch[0, 1].

8.10 PCI Bus Emulation and Port 80 Decode Support

Starting from Skylake, PCIe* RP subtractive decode capability has been removed. The I/O cycles within the 80h – 8Fh range however, can be explicitly claimed by the PCIe* RP by setting “MPC.P8XDE”, PCI offset D8h[26] = 1 (using byte access). The claiming of these cycles is not subjected to standard PCI I/O Base/Limit and I/O Space Enable fields. This allows a POST-card to be connected to the Root Port either directly as a PCI Express device or through a PCI Express to PCI bridge as a PCI card.

Requirement on this support:

1. Any I/O reads or writes will be forwarded to the link as it is. Root Port does not shadow the value written to the device targeting these I/O ranges. The device will need to be able to return the previously written value, on I/O read to these ranges.
2. BIOS must ensure that at any one time, no more than one Root Ports are enabled to claim Port 8xh cycles.
3. Note: BIOS must also configure the corresponding DMI registers GCS.RPR and GCS.RPRDID to enable DMI to forward the Port8x cycles to the corresponding PCIe* RP, refer to section 2.5.5

If Cardbus is connected to the RP, BIOS is required to set PCIBEM, PCI offset ECh[2] = 1. This bit must not be set if P2P bridge is connected instead.

8.11 Sample ASL Code for PME/ Hot Plug Support

ASL sample code is given below to help illustrate the ACPI name space object tree structure layout and location of the ACPI objects involved in the PCI Express* PME and hot plug support.

```
 Scope (\_SB)
{
        Device (PCI0)              // PCI Bus0
  {
    Name (_BBN, 0)         //   Bus Number of PCI0

    Method(_PRT, 0)
        {
        // _PRT package  for PCI0
    }

Method (_OSC, 5 )

//   See 8.8.3 for example
 {
  // Examine input arguments  for OS Capabilities.

  // If OS support PCI Express native hot plug,

// then disable PME/hot plug SCI generation by all PCH root ports
       ……..

Return  (BUF1)
  // Return capabilities info back to OS.
 }
         ……..

        Device (PEX1)
        // PCI Express Root Port 1 on PCH
          {
         Name(_ADR, 0x001C0000)
         // Device28:Func0

         Method(_PRT, 0)
        {
        // return _PRT package for PCI Express root port 1
    }

Name ( _PRW, Package () {0x09,0x03} )// PME Wake capability reporting


Device (REP1)                             // Child device attached to Root Port
1
           {
               Name(_ADR, 0x0)    //  Device0:Func0  (assume  single  function
device)

 Method (_STA, 0 )
 {
       // Return the current device status.
       // Device Presence/Absence can be determined by PDS bit
                                 // of  the Slot Status register at offset 1Ah of
PCI Express
               // Capability structure.
 }


Method (_EJ0, 1 )
 {
    // Method for device eject
 }

   ……

 } // End of device REP1

    } // End of device PEX1
```

```
        Device (PEX2)

        // PCI Express Root Port 2 on PCH
               {
                Name(_ADR, 0x001C0001)

            // Device28:Func1

             Method(_PRT, 0)
        {
        // return _PRT package for PCI Express root port 2
        }

    Name ( _PRW, Package () {0x09,0x03} )    // PME Wake capability reporting

    Device (REP2)

        // Child device attached to Root Port 2
                {

                    Name(_ADR, 0x0)    //Device0:Func0   (assume   single   function
    device)

             Method (_STA, 0 )
        {
            // Return the current device status
        }

            Method (_EJ0, 1 )
        {
            // Method for device eject
        }

        ……

        } // End of device REP2

              } // End of device PEX2


        } // End of device PCI0
    } // End of \_SB scope


Scope (\_GPE)                                                                    //
    GPE event handlers
{
     Method (_L61, 0)

        //  PCI Express Hot Plug event handler
    {

        // Find the source of the hot plug event
        // If the event came from root port1, then
            //        Notify (\_SB.PCI0.PEX1.REP1, event_type)
        // If the event came from root port2, then
            //        Notify (\_SB.PCI0.PEX2.REP2, event_type)
        //  ……..
    }

     Method (_L69, 0)                            //  PCI Express PME event handler
    {
        // Find the source of the PME event
        // If the event came from root port1, then
        // Notify (\_SB.PCI0. PEX1, 0x02) // Notify wake event at root port #1
```

```
        // If the event came from root port2, then
        // Notify (\_SB.PCI0.PEX2, 0x02) // Notify wake event at root port #2
        //  ……..
    }

} // End of \_GPE scope
```

8.12 Non-Common Clock with SSC

The need for an inexpensive PCIe* cabling solution for PCIe* SSDs requires supports of non-common clock mode with spread spectrum enabled. To enable this, ensure flash descriptor PCH Strap depending on which Root Port is selected, correspond to Dxx:F0/F4 + 32Ch [31:28] respectively (e.g. PCIe Controller 1 Port 1 SRIS Enabled).

1. BIOS read the PCIESTS2 register bits [31:28] on Port 0 of each x4 PCIe* instances to check which of the PCIe* root ports have the clocking mode enabled
2. For the ports which have the clocking mode enabled:
   a. Program the PCIe* root port LCAP.APMS to “10b” (combined with step e)
   b. Program the PCIe* root port MPC2.ASPMCOEN, PCI offset D4h[4] to ‘1b’ to instruct that PCIe* root port to use the ASPM Control Override bits instead of the ASPM Control in the Link Control register.
   c. Program the MPC2.ASPMCO, PCI offset D4h[3:2] to “10b” to enable ASPM L1 only. ASPM L0s is disabled.
   d. Program the MPC.UCEL, PCI offset D8h [20:18] to “111b” to advertise more than 4 us L0s exit latency.
      i. Note: It is expected that only new devices which are designed to comprehend the new 6000 ppm will be connected using this new form factor. The new devices must disable ASPM L0s either through the new ASPM Optionality ECN or through proprietary method.
      ii. Worst case CDR lock time on the PCH PCIe* root port receiver is expected to be less than 10 us.
   e. Program LCAP.EL1 to “100b” to comprehend the extended CDR lock time (< 10 us) in the new clocking mode. Program PCI offset 4Ch [17:15, 11:10] to [“100b”, “10b”]. Note: bit22 is also RWO. If its value change is intended, need to be programming in single DW write.
   f. Set the HSIO Elastic Buffer Watermark registers for non-common clock PCIe lane as follows
      i. HSIO_PCS_DWORD8[20:16] to 9h
      ii. HSIO_PCS_DWORD8[12:8] to 18h
   g. Program PCIEALC.BLKDDA, PCI offset 338h [26] to ‘0’ to allow the PCIe* root port to start receiver detection and subsequently train.

8.13 IOSF Port Configuration and Grant Count Programming

The following configurations show the setting of IOSF fabric register (PSF register) based on PCIe* port configuration.

- **PCIe Root Port Configuration**
  - Configuration required for enabled IOSF channel of PCIe root port controller only.
    - x1 lane – GC=1
    - x2 lanes – GC=2
    - x4 lanes – GC=4
- **Device Grant Count Reload Register**
  - PSF_<x>_DEV_GNTCNT_RELOAD_DGCR<GC#>, please refer to Chanel configuration recipe at Table 8-1. Device Grant SKL PCH-LP and PCH-H PCIe IOSF Channels for SKL PCH-H and PCH_LP respectively.

Table 8-1. Device Grant SKL PCH-LP and PCH-H PCIe IOSF Channels

SKL PCH-LP	SKL PCH-H
[PSF1] SPA Channel [0; 1; 2; 3] {5, 6; 8, 9; 11, 12; 14, 15}	[PSF1] SPA Channel [0; 1; 2; 3] {11, 12; 14, 15; 17, 18; 20, 21}
[PSF1] SPB Channel [0; 1; 2; 3] {19, 20; 22, 23; 25, 26; 28, 29}	[PSF1] SPB Channel [0; 1; 2; 3] {24, 25; 27, 28; 30, 31; 33, 34}
[PSF1] SPC Channel [0; 1; 2; 3] {33, 34; 36, 37; 39, 40; 42, 43}	[PSF1] SPC Channel [0; 1; 2; 3] {37, 38; 40, 41; 43, 44; 46, 47}
	[PSF1] SPD Channel [0; 1; 2; 3] {50, 51; 53, 54; 56, 57; 59, 60}
	[PSF1] SPE Channel [0; 1; 2; 3] {63, 64; 66, 67; 69, 70; 72, 73}

Example for SKL PCH-LP on SPA Channel:

- [PSF1] SPA Channel [0; 1; 2; 3] {5, 6; 8, 9; 11, 12; 14, 15}
- SPA Channel 0:
  - PSF_<1>_DEV_GNTCNT_RELOAD_DGCR<5>
  - PSF_<1>_DEV_GNTCNT_RELOAD_DGCR<6>

- **Target Grant Count Reload Register**
  - PSF_<x>_TARGET_GNTCNT_RELOAD_PG<0/1>_TGT<GC#>, please refer to Chanel configuration recipe at Table 8-2. Target Grant SKL PCH-LP and PCH-H PCIe IOSF Channels for SKL PCH-H and PCH_LP respectively.

Table 8-2. Target Grant SKL PCH-LP and PCH-H PCIe IOSF Channels

SKL PCH-LP	SKL PCH-H
[PSF1, PG1] SPA Channel [0; 1; 2; 3] {4; 5; 6; 7}	[PSF1, PG1] SPA Channel [0; 1; 2; 3] {0; 1; 2; 3}
[PSF1, PG1] SPB Channel [0; 1; 2; 3] {12; 13; 14; 15}	[PSF1, PG1] SPB Channel [0; 1; 2; 3] {4; 5; 6; 7}
[PSF1, PG1] SPC Channel [0; 1; 2; 3] {20; 21; 22; 23}	[PSF1, PG1] SPC Channel [0; 1; 2; 3] {12; 13; 14; 15}

SKL PCH-LP	SKL PCH-H
	[PSF1, PG1] SPD Channel [0; 1; 2; 3] {20, 21, 22; 23}
	[PSF1, PG1] SPE Channel [0; 1; 2; 3] {28, 29, 30, 31}

Example for SKL PCH-LP on SPA Channel:

- [PSF1, PG1] SPA Channel [0; 1; 2; 3] {4; 5; 6; 7}
- SPA Channel 0:
  - PSF_<1>_TARGET_GNTCNT_RELOAD_PG<1>_TGT<4>

8.14 PCI Express* Gen 3 Support

The link equalization procedure is required for components to adjust the transmitter and the receiver setup of each lane to improve the signal quality and meet the requirements of 8.0 GT/s data rate. Please see PCI express 3.0 base specification for more details.
In phase three of the equalization procedure the upstream port transmitter setting is being adjusted against the downstream port receiver setting independently for each lane. The Upstream Port advertises the transmitter coefficients and the preset it is using. The Downstream Port receiver evaluates the setting and may request different preset or coefficient settings until it arrives at the best setting for operating the upstream lanes.

PCH does not support phase three of equalization in hardware therefore alternative software method must be used.
BIOS will enable remote transmitter preset or coefficient override mode to force the upstream port setting using fixed value per lane based on the physical channel characteristinc (static equalization) or sweep thru a range of preset or coefficient evaluating link margins to pick the best setting automatically (software equalization).

Note: PCIe Gen3 ASPM L0s is not supported and must be disabled.

8.14.1 Static Equalization Method

Static equalization assumes fixed coefficients or preset are used by the upstream port transmitter by the means of remote transmitter coefficient override mechanism being enabled during phase 3 of Gen3 link equalization. Coefficients are selected per PCIe lane based on the channel trace length and/or characterization. See EDS for more details on coefficient selection.

Programming steps required to enable static equalization per root port, prior to the first training to Gen3:

1. Ensure link is in Gen1.
2. Program Dxx:Fn + 450h [16,15] = [1,1] to enable remote coefficient override.
3. Set Dxx:Fn + 454h[31] = 1 to enable coefficient mode or clear it for preset mode.

4. Program coefficients for PCIe lanes assigned to the root port in the register fields according to Table 8-3. Remote Transmitter Preset Override Mode Settings or Table 8-4. Remote Transmitter Coefficient Override Mode Settings respectively.
5. Program target link speed by setting LCTL2.TLS to 3.
6. Set LCTL3.PE and perform link retrain by setting LCTL.RL according to PCI Express base specification.

Table 8-3. Remote Transmitter Preset Override Mode Settings

Lane	Preset
Lane 0	454h [5:0]
Lane 1	454h [11:6]
Lane 2	454h [17:12]
Lane 3	454h [23:18]

Table 8-4. Remote Transmitter Coefficient Override Mode Settings

Lane	Pre-cursor	Post-cursor
Lane 0	454h [5:0]	454h [11:6]
Lane 1	454h [17:12]	454h [23:18]
Lane 2	454h [29:24]	458h [5:0]
Lane 3	458h [11:6]	458h [17:12]

8.14.2 Software Equalization Method

Software equalization method will iterate over a list of preset or coefficients using remote transmitter setting override mode and perform evaluation of each of the settings by inserting jitter and retrieving the number of framing errors at given jitter level for each lane.

Software equalization flow requires access to the HSIO lane specific registers using register based sideband access. Refer to section 2.2.1 for sideband access method and to EDS volume 2 for HSIO register details and lane addressing.

Perform the evaluation for each upstream setting, with the following flow:

1. Ensure link is in Gen1
2. Configure HSIO lane for manul jitter insertion by clearing HSIO_RX_DWORD8[10]
3. Set the remote transmitter setting override to the first/next setting for all lanes as in steps 1-3 of static equalization method.
4. Set LCTL3.PE, up-train the link to Gen3 and make sure link is in L0 state
5. Reset hardware error counters by clearing and then setting Dxx:Fn + 450h [23]
6. Wait 5us
7. If recovery entry count located at Dxx:Fn + 450h [31:24] is not zero run the flow for next upstream port setting
8. For each jitter level N from in range 0 to 32
   8.1. Freeze lane calibration by setting HSIO_RX_DWORD23[4] and also HSIO_RX_DWORD23[2] for PCH-LP C0 and PCH-H D0 and newer, set

HSIO_RX_DWORD9[30,28,26,24] to 1111b and then toggle HSIO_RX_DWORD25[3] by clearing it and setting again.

8.2. Enable jitter insertion module at level N for all lanes that did not hit the framing error threshold

8.2.1. Set HSIO_RX_DWORD60 to 73010100h OR N and clear HSIO_RX_DWORD57[21,20].

8.2.2. Set HSIO_RX_DWORD59 to 4

8.2.3. Set HSIO_RX_DWORD57[30,29,28,0] to 0110b

8.2.4. Toggle HSIO_RX_DWORD56[16] by clearing it and setting again

8.3. Reset hardware error counters

8.4. Wait for 500us

8.5. Disable jitter insertion module by clearing HSIO_RX_DWORD57[29] and toggle HSIO_RX_DWORD56[16] by clearing it and setting again

8.6. Un-freeze lane calibration by clearing HSIO_RX_DWORD23[4] and also HSIO_RX_DWORD23[2] for PCH-LP C0 and PCH-H D0 and newer, clear HSIO_RX_DWORD9[30,28,26,24] and then toggle HSIO_RX_DWORD25[3] by clearing it and setting again.

8.7. Read framing error counts for each lane by setting dword Dxx:Fn + 480h to the lane index (0 to 4) and read the respective count from Dxx:Fn + 480h [31:8]

8.8. If the link downtrained or recovery entry count exceeded threshold of 40 assume margin for all lanes that have not completed margining is N and proceed to next upstream setting

8.9. If error count for given lane exceeded the threshold of 4 the lane margin at current setting is N and the lane margining completed.

8.10. Update the best setting for given lane if the lane margin is greater than for any of the previous settings

9. Reiterate from 1 if there are more upstream settings to evaluate
10. Apply the best setting for each lane same as in static equalization method

For some devices Gen3 rate cannot be sustained for sufficient period without vendor specific configuration, therefore if after completing the above flow link does not train to Gen3, the following fallback flow is recommended:

1. Reset the device using a dedicated GPIO (see Device Reset below for detailed device reset flow)
2. Disable equalization phase two and three by setting Dxx:Fn + 450h [18, 17] to 11b
3. Perform the software equalization flow as above with the following modifications:
   a) Use Upstream Port Transmitter Preset field in the Lane Equalization Control Register for each lane inside the Secondary PCI Express Extended Capability to indicate the upstream preset (do not use the registers from the Static Equalization Method)
   b) If the link does not train to Gen3 at given preset for the first time reset the device and try the same preset again
4. Re-enable equalization phase two and three (clear Dxx:Fn + 450h [18, 17])
5. Apply the best preset for each lane same as in static equalization method

8.14.2.1 Device Reset

During the software equalization it might be necessary to reset the device using PERST# signal with dedicated GPIO pin as indicated above. The following is the recommended flow for resetting the device:

1. Set LCTL.LD to 1
2. Poll link sate located at Dxx:Fn + 328h [31:24] until equal 5Eh or 60h
3. Set Dxx:Fn + 338h [26] to 1
4. Clear LCTL.LD
5. Poll link sate located at Dxx:Fn + 328h [31:24] until equal 01h
6. Assert device reset (PERST#)
7. Wait 100us
8. De-assert device reset (PERST#)
9. Clear Dxx:Fn + 338h [26]

8.15 Additional PCI Express* Programming Steps

BIOS is required to perform the additional steps listed below prior to PCIe* enumeration.

1. Refer to section 29.1 for function disable requirement due to lane ownership and clkreq mapping
2. Refer section 8.2 for Root Port configuration.
3. Refer to section 8.12 for NCC with SSC programming support.
4. If not NCC config, set Active State Link PM Support (LCAP.APMS) = '11b'
5. Refer to section 8.2.11 for peer memory access programming requirement.

For first PCIe RP in each controller set Dxx:Fn + 338h[28,29] = 11b

Power Management-related programming:

6. Set Dxx:Fn + 478h[31:24] = 28h (Gen3 Active State L0s Preparation Latency)
7. Set Dxx:Fn + 318h [31:16] = 1414h (Gen2 and Gen1 Active State L0s Preparation Latency)
8. Enable clock gating by
   a. Program first RP in each controller PCI offset E1h [7, 5, 4, 2] = [1, 1, 1, 1] using sideband access
   b. Program Dxx:Fn + E1h [1, 0] = [1, 1]
   c. Program Dxx:Fn + F7h [6] = 1
9. Enable squelch by
   a. Program Dxx:Fn + 324h[5] = [1]
   b. Program Dxx:Fn + E8h[1] = [1]
10. Enable squelch for GEN3 capable PCIe ports by set PCI offset 324[7] = 1b
11. Enable PLL shutdown for the first port in each controller using sideband access
    a. Set Dxx:Fn + F5h[1, 0] = [1, 1]
    b. Program Dxx:Fn + E8h[17,15] = [1,1]
12. Dedicated clkreq per port related programming (always true for mobile system, assumed to be true for desktop)
    a. Program Dxx:Fn + 424h [7, 5, 4] = [1, 1, 1]

b. Program first RP in each controller PCI offset 428h [5, 0] = [1, 0] using sideband access

c. Other related programming requirement are stated in section 8.3

13. If PXP PLL Shutdown is enabled (if step 11 is executed),

    a. Set L1 Exit Latency, Dxx:Fn + 4Ch [17:15] = 100b
    Else set L1 Exit Latency, Dxx:Fn + 4Ch [17:15]= 010b

14. Configure PCI Express Number of Fast Training Sequence (PCIENFTS) for each port (B0:Dxx:Fn):

    a. Set Dxx:Fn + 314h [31:24, 23:16, 15:8, 7:0] to [7Eh,70h, 3Fh, 38h]

    b. Set Dxx:Fn + 478h [31:24, 15:8, 7:0] to [28h, 3Dh, 2Ch]

15. Set “Poisoned TLP Non-Fatal Advisory Error Enable”, Dxx:Fn + D4h[12] to 1

16. Set “Link Speed Training Policy”, Dxx:Fn + D4h[6] to 1. Make sure this is after the mod-PHY related programming completed.

17. Set Common Clock Exit Latency, Dxx:Fn + D8h[17:15] = 4h by setting bit [15] to 1 using byte/word access, the default of bit [17:16] is 01b. This is to avoid touching RWO bit in bit [23]

18. Set MPC.IRRCE, Dxx:Fn + D8h[25] = ‘1b’ using byte access

19. For systems that support MCTP over PCIE set
    MCTP Message Bus Number Check Enable (MPC.MMBNCE) = ‘1b’
    MCTP Support Enable (MPC.MCTPSE) = ‘1b’

20. Configure Transmit Timer settings:

    a. Set PHYCTL2.TDFT, Dxx:Fn + F5h[7:6] = ‘00b’

    b. Set PHYCTL2.TXCFGCHGWAIT, Dxx:Fn + F5h[5:4] = ‘00b’

21. Set IOSF Sideband Interface Idle Counter, first RP in each controller + F7h[3, 2] = ‘00b’ using sideband access.

22. Enable “PME_TO Time-Out Policy”, Dxx:Fn + E2h[6] = ‘1b’

23. Enable “Completion Time-Out Non-Fatal Advisory Error”, Dxx:Fn + 324h[14] = ‘1b’

24. Set “ASPM Optionality Compliance”, Dxx:Fn + 4Ch[22] = ‘1b’, done as part of the step 13, default value is being used.

25. Configure Transmitter Preset for each Upstream and Downstream Port Lane:

    a. Set L01EC.DPL0TP, Dxx:Fn + 22Ch[3:0] = ‘0111b’

    b. Set L01EC.UPL0TP, Dxx:Fn + 22Ch[11:8] = ‘0101b’

    c. Set L01EC.DPL1TP, Dxx:Fn + 22Ch[19:16] = ‘0111b’

    d. Set L01EC.UPL1TP, Dxx:Fn + 22Ch[27:24] = ‘0101b’

    e. Set L23EC.DPL2TP, Dxx:Fn + 230h[3:0] = ‘0111b’

    f. Set L23EC.UPL2TP, Dxx:Fn + 230h[11:8] = ‘0101b’

    g. Set L23EC.DPL3TP, Dxx:Fn + 230h[19:16] = ‘0111b’

    h. Set L23EC.UPL3TP, Dxx:Fn + 230h[27:24] = ‘0101b’

Note: The values above are temporary. Final values are pending on characterization.

26. Enable EQ TS2 in Recovery Receiver Config, Dxx:Fn + 450h[7]= ‘1b’

27. Set “Link Clock Domain Squelch Exit Debounce Timers”, Dxx:Fn + 324[25:24] = ‘00b’. Can combine with step 9

28. If there is no IOAPIC behind the root port, set EOI Forwarding Disable bit (MPC2.EOIFD) to '1b'
    a. Note: if there is IOAPIC behind the root port, no effort done to enable the IOAPIC in client config.
29. System BIOS should initiate link retrain for all slots that has card populated after register configuration.
30. If Transmitter Half Swing is enabled and port is NOT Gen3 capable, program the following sequence
    a. Ensure that the link is in L0.
    b. Program the Link Disable bit (PCI offset 50h[4]) to 1b.
    c. Program the Analog PHY Transmitter Voltage Swing bit (PCI offset E8h[13]) to set the transmitter swing to half/full swing
    d. Program the Link Disable bit (PCI offset 50h[4]) to 0b.
31. Prior to 3rd party code running,
    a. Set MPC.SRL, Dxx:Fn + D8h[23] to 1
    b. Set PCD.SRL, PCR[spx] + 0h[31] to 1 (spx – every port id for PCIe* controllers)

Note: Detection of Non-Complaint PCI Express Devices

Some PCI Express devices do not follow PCI Express Specification and currently report the incorrect Gen capability or link width. This may cause the improper detection of the card by the PCH PCI Express root port. It is recommended to statically force the particular root port that supporting such devices to lower speed.

If the dynamic detection is desired, the following settings may improve the ability of the PCH PCI Express root port to detect these non-compliant PCI Express devices.

If BIOS cannot detect or train the device:

1. Poll Dxx:Fn:328h [31:24] for value <= 32h up to 5 retries.
2. If polling at step #1 is true, Set Dxx:Fn:70h [3:0]= 1h else link is trained and should exit this flow.
3. Wait 100 ms for link to train up
4. Repeat step #1 and if Dxx:Fn:328h [31:24] remained<=32h, set Dxx:Fn:338h [26] to 1b and disable the port using function disable.

Note: The above setting is "as-is" as Intel cannot verify all non-compliant devices. You need to ensure that the workaround works with devices you are planning to use.

8.15.1 Power Optimizer Configuration

1. Enable support Latency Tolerance Reporting (LTR)
    a. Based on the flow diagram shown in Figure 8-8, BIOS will determine the override value (snoop and/or non-snoop) to be programmed in B0:Dxx:Fn + 400h
        i. 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.

ii. Program B0:Dxx:Fn + 64h [11] = 1b

iii. Program B0:Dxx:Fn + 68h [10] = 1b

iv. Program B0:Dxx:Fn + D0h [13:12] = 01b

2. Disable Optimized Buffer Flush/Fill (OBFF)

- Program B0:Dxx:Fx + 64h [19:18] = 0h

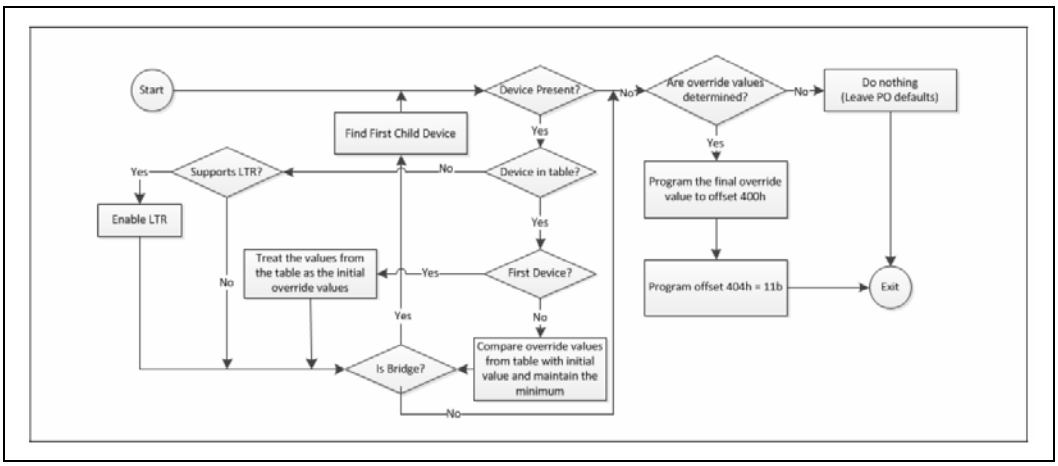
3. If Endpoint device supports LTR, Device Capabilities 2 Register Offset 24h [11] = 1b,
   a. Program Endpoint LTR Mechanism Enable, Device Control 2 Register Offset 28h [10] = 1b when device supports LTR but is not found in override table (table listing correct latency requirements for devices that 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
      - 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 8-8: BIOS Flow for PCIe* LTR Discovery and Enumeration



§ §

9 Intel® High Definition Audio (Intel® HD Audio) Controller

The Intel® High Definition Audio (Intel® HD Audio) controller (Device #31, Function #3) is a PCH internal PCI Express* endpoint device. 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, etc.

This chapter describes BIOS requirements for Intel HD Audio controller support.

9.1 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, etc. 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

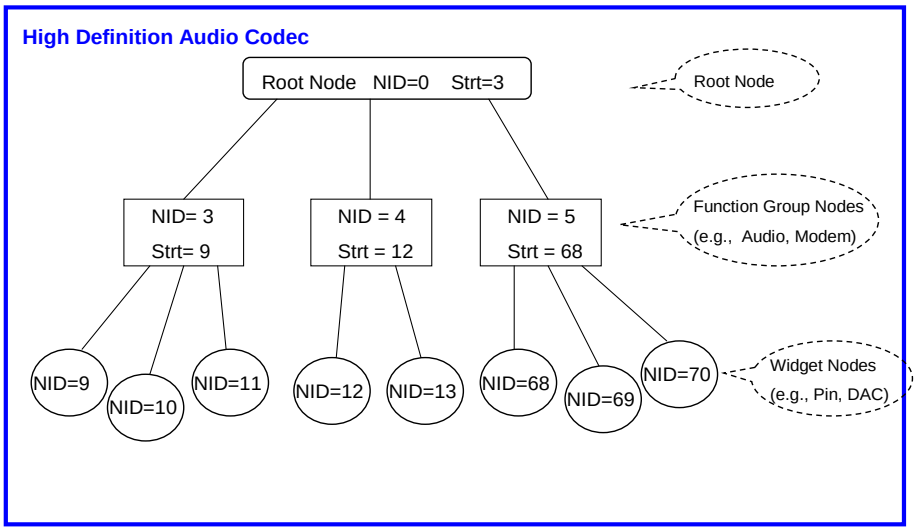
After System BIOS has determined the presence of Intel HD Audio codec(s), it must follow the programming sequence given in this section to update the codec with correct jack information specific to the platform for the Intel HD Audio driver to retrieve and use later. If the codecs are not initialized 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.

9.1.1 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 9-1.

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 SDI0 (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 9-1: Intel® HD Audio Codec Node Structure and Addressing

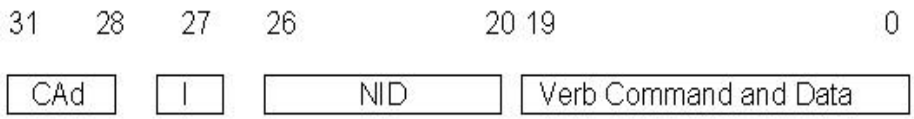


- High Definition Audio Codec Node Structure and Addressing-

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. See the Intel HD Audio Specification for details.

9.1.2 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 System BIOS. The Codec Verb Tables are based on codec specific information (coded datasheet) and platform design specific information (schematics) and are built by System BIOS 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
```

9.1.3 Codec Initialization Programming Sequence

After System BIOS 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.

System BIOS should ensure that the Intel HD Audio HDBAR HDA PCI:10-17h contains a valid address value and is enabled by setting HDA PCI:04h[1]. System BIOS must ensure program as mentioned in section 9.5, and then 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, please contact your codec vendor for detail. When clearing this bit and setting it afterward, BIOS must ensure that minimum link timing requirements (minimum RESET# assertion time, etc.) are met.

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

1. Read the VendorID/DeviceID pair from the attached codec.
   - Verify that the ICB bit, HDBAR + 68h[0], is 0.

- Write verb C00F0000h (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. BIOS 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.

2. 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 System BIOS implementation-specific steps and can be done in different ways. If a System BIOS has prior knowledge of a fixed platform/codec combination (for example,, for a System BIOS 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 a System BIOS to support multiple codec/platform combinations, an internal match-list might be needed to match a platform/codec combination to a codec verb table.

3. 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. BIOS 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.

9.1.3.1 Codec Initialization Sample Code

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

```
;-----------------------------------------------------------------------------
;
; Procedure:   InitializeHigh 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                        ; Is the codec address correct?
        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

FPSupportDone:

        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:
```

9.1.4 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 System BIOS 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 System BIOS 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 System BIOS 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 9.5.
2. When HD-A bus link is configured for 6MHz
   a. Set HDBAR + 08h[0] to take the controller out of reset, wait for about 1ms.
   b. Set HDA PCI offset 54h[1:0] to put controller into D3.
3. Program Verb Tables to codecs as BIOS did during the POST.

9.2 Intel® HD Audio Controller Configuration

Once the Intel HD Audio codec is determined to be present and the Intel HD Audio controller is, System BIOS 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:2Ch~2Eh to OEM-specific IDs. This is similar to the SID/SVIDs given to other PCH devices such as IDE, SATA, SMBus, USBs, etc.

9.3 Intel® HD Audio PME Event

Although it is a PCI Express* Root Complex Integrated endpoint, the Intel HD Audio controller in PCH is not capable of supporting the native PME software model. Its PME is supported in the same manner as a PCI PME.

The PME_B0_STS/PME_B0_EN bit pair in PCH GPE0 register (ABASE + 20h[13], ABASE + 28h[13]) is shared between all PCH bus 0 devices, including Intel HD Audio. To support this PME feature in an ACPI OS environment, System BIOS needs to provide the proper _PRW object and \GPE._L6D() control method in the ACPI name space.

9.4 Intel® HD Audio Isochronous ("Isoch") Support on VC1

PCH can be configured to enable isoch service for the Intel HD Audio controller. This involves virtual channel configuration and port arbitration table initialization. In the absence of a PCI Express*-aware OS, System BIOS can enable isoch support for Intel HD Audio to take advantage of this PCI Express* performance feature.

Intel HD Audio and DMI both support virtual channel VC1 in addition to the required (default) VC0. It is necessary for System BIOS to configure and enable VC1 on both Intel HD Audio and on DMI link as the basis of isoch support for Intel HD Audio. The following sections describe the configuration requirements and programming sequences of enabling VC1 for DMI port and Intel HD Audio. The configuration programming should occur before standard PCI enumeration unless stated otherwise.

Note: The Processor SA needs to be configured properly as well for VC1 traffic to go through. Please refer to the appropriate Processor SA BIOS Writer's Guide for details.

9.4.1 DMI VC1 Configuration

The VC1 virtual channel on DMI link must be configured properly by software before any VC1 traffic can go through. For Intel HD Audio isoch support, System BIOS should follow section 7.2.4 to configure and enable VC1 on DMI link.

Note: VC1 is optional for Intel HD Audio.

9.4.2 Intel® HD Audio VC1 Configuration

1. Permit non snoop on VC0, HDA PCI offset C4h[24] = 1b.
2. Enable VC1 traffic, TTCCFG.HCDT, HDA PCI offset E4h[1] = 1

9.5 Intel iDisplay Audio Link

The Intel iDisplay Audio link is a new link between the PCH and iDisplay Audio codecs (which could be either a processor or a standalone analog audio codec). It allows the Intel HD Audio controller in the PCH to output audio streams to HDMI/DP and WiDi/WiGig via the processor's Display interfaces. Similarly, it provides the capability for audio input streams to be received from the processor. Note however that within the 2015 timeframe, the processor may not be taking advantage of this input audio path from the Display. This feature is specific to the Skylake/Kaby Lake PCH implementation. It is not required by PCH which are paired with processors which don't support the iDisp Audio Link.

Skylake/Kaby Lake codec can be addressed from PCH by using CAd==2.

For the platform does not support Intel Display Audio, BIOS needs to disconnect SDI2 signal to prevent iDisplay Audio codec enumeration by set PCR[DSP] + 534h [2] = 1b, and does not perform device initialization; this can prevent Intel Display Audio codec present in OS.

9.5.1 Intel iDisplay Audio Verb Table

Below is an example for Intel iDisplay Audio Verb Table, BIOS is responsible to load verb table during POST.

Verb table contains valid pin configuration that reflects platform configuration with HDMI/DP ports; the pins not connected externally must be disabled.

Note: Platform design includes PortB, then vanilla mode must be enabled.

Platform design does not include PortB, then vanilla mode must be disabled.

```
HDAUDIO_VERB_TABLE HdaVerbTableDisplayAudio = {
  //
  //  VerbTable:
  //  Revision ID = 0xFF
  //  Codec Vendor: 0x80862809
  //
  {
    0x8086,     // Vendor ID
    0x2809,     // Device ID
```

```
    0xFF,       // Revision ID (any)

    0xFF,       // SDI number (any)

    3 * 4 + 2   // Data size in DWORDs – To disable Vallina mode. "2" must be changed
to "1", and remove 0x00878100 in the end of table.

  },

  {

    //

    // Display Audio Verb Table

    //

    // Enable the third converter and Pin first (NID 08h) – Disable Vanilla mode

    0x00878101,

    // Pin Widget 5 - PORT B - Configuration Default: 0x18560010

    0x00571C10,

    0x00571D00,

    0x00571E56,

    0x00571F18,  // Enable PORT-B by set to "18"

    // Pin Widget 6 - PORT C - Configuration Default: 0x18560020

    0x00671C20,

    0x00671D00,

    0x00671E56,

    0x00671F18,  // Enable PORT-C by set to "18"

    // Pin Widget 7 - PORT D - Configuration Default: 0x18560030

    0x00771C30,

    0x00771D00,

    0x00771E56,

    0x00771F58,  // Disable PORT-D by set to "58"

    // Disable the third converter and third Pin (NID 08h) – Enable Vanilla mode

    0x00878100

   // To disable Vanilla mode, above line must be removed, and change data size in table
header.

  }

};
```

9.6 Additional Intel® HD Audio Programming Steps

9.6.1 PCI Mode

HDA should be operating under PCI mode. IP default is using PCI mode, just need additional PSF register programming below:

1. Set PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[HDA][19, 18] to [1, 1]

9.6.2 Function Disabling Steps

To disable HDA (inclusive of Intel® Smart Sound Technology)

1. Set HDA PCI offset 54h[1:0] = 1b, 1b
2. Set PWRMBASE + 628h [23] = 1b
3. Set FNCFG, PCR[DSP] + 530h [3] = 0b
4. Set FNCFG, PCR[DSP] + 530h[5, 4, 2, 0] = 0b, 1b, 1b, 1b
5. Set PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[HDA][8] = 1b

To disable Intel® Smart Sound Technology only:

1. Set FNCFG, PCR[DSP] + 530h[5, 4, 3, 2] = 0b, 1b, 0b, 1b

Note: Because the BCLD, bit 5 of FNCFG is R19.5WO and it impacts other register field programmability, the disabling step for the Intel Smart Sound Technology particularly, has to be done at later stage, after HDA init step that might has dependency on it is done.

9.6.3 Misc 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. BIOS is required to program with the expected setting before controller is out of reset (see 9.6.4).
   a. APLLP0.KPKISC, PCR[DSP] + 610h [31:30] = 11b
   b. APLLP0.KIS, PCR[DSP] + 610h [29:28] = 11b
   c. APLLP0.KPS, PCR[DSP] + 610h [27:26] = 11b
   d. APLLP0.PLLWC, PCR[DSP] + 610h [22:16] = 1Eh
   e. APLLP1.DCOTC, PCR[DSP] + 614h [13:8] = 1Eh
   f. APLLP2.KPCE, PCR[DSP] + 618h [11:8] = 1h
   g. APLLP2.KICE, PCR[DSP] + 618h [4:0] = 1Dh
3. Program CGCTL.SROTCGE, HDA PCI offset 48h[18] = 0b

4. Per BIOS policy, program the “Ownership Selection”, PCR[DSP] + 61Ch[9:8], according to platform configuration (Azalia vs I2S). This is only supported on PCH-LP platform.
   a. HD-A Link
      i. Set PCR[DSP] + 61Ch[9:8] = 00b
   b. HD-A Link and I2S Port (Shared)
      i. Set PCR[DSP] + 61Ch[9:8] = 01b
   c. I2S Ports
      i. PCR[DSP] + 61Ch[9:8] = 11b
      ii. HDBAR + 08h[0] = 1b to deassert the HDA controller RESET#
5. Program HDA PCI offset C4h[27:26] = 01b to set Buffer Size Minimum Threshold to 128B.
6. Program HDA PCI offset C8h[21:20] = 10b to set Input Stream L1 Exit Threshold 2 to 1/8 of FIFO size
7. Program HDA PCI offset D0h[21:20] = 11b to set Output Stream L1 Exit Threshold 2 to 1/4 of FIFO size
8. Set HDBAR + 1048h[6:0] = 29h.

9.6.4 Static Frequency Switching

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

1. If iGFx is to be disabled,
   a. set iDisplay Link’s LCTLx.SPA to 0
   b. skip all other iDisplay Link related steps
2. 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).

   Set the following registers before iDisplay Link is brought out from reset (LCTLx.SPA = 0b, poll LCTLx.CPA = 0b):

   a. Set SEM1.TMODE, HDA PCI offset C0h [12] = 0b for 2T or 1b for 1T Mode
   b. Set T mode in the display codec using iGFx AUD_FREQ_CNTRL MMIO register access:

      GttMmAdr + 65900h[15] = 0b for 2T or 1b for 1T Mode
   c. Set iDisplay Link freq using iGFx AUD_FREQ_CNTRL MMIO register access, GttMmAdr + 65900h[4:3]:

      96 MHz: 10b

      48 MHz: 01b
3. 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):

a. OUTPAY: HDABAR + 04h[15:0]

b. INPAY: HDABAR + 06h[15:0]

accordingly:

24 MHz:

OutputPayloadWords = 3Ch (60 words of payload)

InputPayloadWords = 1Ch (29 words of payload)

12 MHz:

OutputPayloadWords = 1Ch (28 words of payload)

InputPayloadWords = 0Dh (13 words of payload)

6 MHz:

OutputPayloadWords = 0Dh (13 words of payload)

InputPayloadWords = 05h (5 words of payload)

4. For each link:

   a. Platform will determine the optimal supported link frequency and passed in as BIOS policy

   b. Check if connected codec supports required BCLK frequency by sending codec command with Verb ID = F00h, Parameter ID = 16h

   c. Turn off the link: set LCTLx.SPA to 0, poll LCTLx.CPA till it is 0

   d. Set LCTLx.SCF accordingly

   e. Turn on the link: set LCTLx.SPA to 1, poll LCTLx.CPA till it is 1

5. Wait 512us

6. For non-S3 path, for each link, get current frequency from all codec by sending codec command with Verb ID F37h: GET_CCF

   a. If any of the results does not match the desired frequency, perform the below step to revert back to its default link frequency:

      i. Set LCTLx.SPA to 0

      ii. Poll LCTLx.CPA till it is 0

      iii. Set LCTLx.SCF back to default frequency

      iv. Set LCTLx.SPA to 1

9.6.5 Registers Lock Down and Related Field

1. Set PCR[DSP] + 530h[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.

9.6.6 Power Management and Clock Gating Requirement

1. For B0 stepping onward,
   a. HDABAR + 104Bh[5,3] = 11b to allow power gating assert and sleep enable.
   b. HDABAR + 104Bh[2,1,0] = 000b
2. Set PCR[DSP] + 530h[5] = 0b to enable power gating
3. Set PCR[DSP] + 530h[3] = 0b to enable clock gating

10 Gigabit Ethernet Controller

PCH integrates a Gigabit Ethernet Controller which connects to a Platform LAN Connect device over SMBus and any one of the PCH's PCI Express* root ports. The PCIe* port is selected through a soft strap. See EDS for supported Platform LAN Connect devices.

The GbE Controller is enabled by default. SPI flash must be programmed with a valid descriptor and GbE NVM image in a system where the GbE Controller is enabled.

The GbE Controller functionality can be shared with certain PCI Express* Root Port. The PCI Express* Root Port used by GbE needs to be disabled, refer to section 5.3 and section 29.1

10.1 GbE Controller Support

System BIOS could include the PXE Option ROM into the final BIOS image. The PXE option ROM will be given control during POST. The PXE will enable the hardware for network boots. The System BIOS writer should use the utility provided with the PXE Option ROM to change the PCI Device ID field that is appropriate to the platform LAN currently being used. By running this utility the PCI Device ID in the PCI Option ROM header field gets changed.

Please contact your Intel Representative for PXE Option ROM code and utility.

10.2 Enabling / Disabling the Internal GbE Controller

In PCH systems, changing the internal GbE controller from disabled to enabled during POST requires a system reset (IO port CF9h = 0Eh) immediately after clearing the LAN disable (Static PG) bit in the PWRMBASE + 620h [0]. If GbE supported Intel ME is enabled and the LAN disable bit PWRMBASE + 620h [0] has changed, then system BIOS must set PMC PCI offset 0ACh[20] prior to issuing a platform reset (IO port CF9h = 6h or Eh).

10.2.1 Enable the Internal GbE Controller

System BIOS can use the following procedure to enable GbE controller.

1. Set PWRMBASE + 620h [0] = 0b to enable GbE controller.
2. Read PWRMBASE + 620h [0] value.
3. Set "LOCKLANDIS", GbE PCI config + A4h bit[0] to 1
4. Issue global reset if Intel ME is enabled or power cycle reset when Intel ME is not enabled.
5. Cross reference to section 5.3.1 where all static PG'ed controller are being handled at once.

10.2.2 Disable the Internal GbE Controller

The Internal GbE Controller should be disabled under the following conditions:

- SPI is being used but GbE flash region does not exist.
- User requesting to disable the Integrated LAN controller.
- GbE does not own any lane, refer to section 29.1

System BIOS should perform the following procedure to disable the Internal GbE Controller:

- Prior to any PCI Configuration accesses to GbE,
  1. If Intel ME enable then detect if it supports GBe
     a. Read FWSM_S[15] bit in MBARA + 5B54h register.
  2. Set PWRMBASE + 620h [0] = 1b to disable GbE Controller.
  3. Set “LANDISCTRL”, GbE PCI config + A0h bit[0] to 1
  4. Set “LOCKLANDIS”, GbE PCI config + A4h bit[0] to 1
  5. Issue global reset if FWSM_S[15] = 1b in step 1a above if Intel ME is enabled or a power cycle reset when Intel ME is not enabled.

10.3 Accessing LAN MAC address Recommendation

After power-on or system reset, the LAN will reads the flash and programs its first Receive Address Register pair with the MAC address. System BIOS should be able to read these two controller registers and retrieve the MAC address that was in the flash. The first Receive Address Register pair is at "GbE CSR base address + 5400h” and “GbE CSR base address + 5404h”.

System bios can implement the below procedures to retrieve GbE LAN MAC address (i.e. standard MAC address) after PCI enumeration has completed:

Note: BIOS must use DWORD accesses to read or write the LAN controller registers.

1. Read the GbE CSR base address from PCI BAR 0 of the LAN
2. Read the first and second MAC address words from the DWORD at (GbE CSR base address + 5400h).
3. Read the third MAC address word from the DWORD at (GbE CSR base address + 5404h)
4. Register value at 5400h and 5404h are respectively the first 4 bytes and the last 2 bytes of the MAC address

Note: Those values need to apply byte swapped to get correct MAC address. Thus, it means the low byte of the 32-bit value is the first byte of the MAC address for both 5400h and 5404h.

5. Make sure that ‘bit 31’ is set to ‘1’ on the value read from (GbE CSR base address + 5404h). If ‘bit 31’ is set to ‘0’, the address is not valid and should not be used

Note: Refer to section 13.4.2 and 13.4.3 of “PCI Express* GbE Controllers Open Source Software Developer's Manual” for more details.

10.4 Wake-On-LAN (WOL) Implementation

To support WOL after an S0 state with the Internal GbE Controller, System BIOS must implement the following:

- System BIOS must enable the generation of an SMI when the SLP_EN bit of the Power Management 1 Control register is written (ABASE + 04h[13]). This is accomplished by setting SLP_SMI_EN bit of the SMI Control and Enable register (ABASE + 30h[4]).
  1. Clear PMC PCI offset A2h[14] = 0b to ensure the LAN PHY will be powered for WOL when the power source is either the AC or the DC battery.
  2. Clear PMC PCI offset A2h[13] = 0b to ensure the LAN PHY will be powered for WOL in Deep Sx.
  3. Set PMC PCI offset A2h[12] = 1b to ensure the LAN PHY will be powered for WOL after a G3 transition.
  4. Set PMC PCI offset A2h[11] = 1b to ensure the LAN PHY will be powered for WOL from Sx.
  5. “PME_B0_EN”, ABASE + 28h[13], bit must be programmed to enable WOL from Sx at the Power Management Controller.
  6. Optionally BIOS can set WOL Enable Override, PMC PCI offset A4h[13], bit to guarantee the WOL is enabled at the Power Management Controller even in surprise S5 cases, such as power loss and Power Button Override.
  7. Moreover, system BIOS also require to enables in the LAN device by performing the WOL configuration requirements in the GbE region of the SPI flash.

System BIOS or OEMs are required to configure descriptor and GbE image in the SPI flash properly to enable WOL. Please refer to PCH SPI programming Guide and relevant LAN datasheet for more details.

10.5 Additional GbE Controller Configurations for WOL Support

System BIOS requires to program the registers listed below for internal GbE.

1. Set MBARA + 2Ch[31] = 1b
2. If WOL is enabled set MBARA + 2Ch[30] = 1b
   Else if disabled set MBARA + 2Ch [30] = 0b

Note: The settings above should be done via a read-modify-write

As part of the LAN enumeration cycle (init, warm reset, platform reset, resume) the BIOS should also set the LAN BAR0.FEXT[31:30]

System BIOS requires to program the registers listed below for internal GbE to function upon S0 to S3/S4/S5 transition (when GbE device is in D0 state).

1. If MBARA + 5800h[0] = 1b then proceed the steps below else skip remaining steps.
2. System BIOS perform read to MBARA + 5400h[31:0], MBARA + 5404h[31:0] and MBARA + F00h[31:0].
3. Ensure MBARA + F00h[5] = 1b

   a. Set MBARA + F00h[31:0] value with the value read in step 2 OR with value 20h (set bit 5).
   b. Read MBARA + F00h.
   c. If MBARA + F00h [5] = 1b (true) continue else wait X Sec and go back to step 3.b for Y times (X*Y totals to ~200ms) if false – exit flow by jumping to step 32.
4. If MBARA + 5B54h[15] = 1b, then jump to step 10.
5. If MBARA + F10h[2] = 1b, then set MBARA + F10h[1] = 1b. Else clear MBARA + F10h[1] = 0b
6. MBARA + 20h = 043f0000h, Verify MBARA + 20h[28] = 1b
7. Wait 4 mSec
8. Set MBARA + 20h = 04390000h or with 400h or with 40h if MBARA + F10h[3] = 1b or with 04h if MBARA + F10h[2] = 1b.
9. Verify MBARA + 20h[28] = 1b
10. Set MBARA + 20h = 043F6400h
11. Wait 4 ms.
12. Set MBARA + F10h[6] = 1b (Read-Modify-Write).
13. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 4310010h
14. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 4320000h or with the least significant word of MBARA + 5400h that read in step 2.
15. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 4310011h
16. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 4320000h or with the most significant word of MBARA + 5400h that read in step 2.
17. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 4310012h
18. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 4320000h or with the least significant word of MBARA + 5404h that read in step 2
19. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 4310013h
20. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 4328000h
21. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 4310001h
22. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 8320000h
23. Verify MBARA + 20h[28] = 1b, TEMP[15:0] = MBARA + 20h[15:0]
24. Set MBARA + 20h = 4320000h or TEMP[15:0] or 0001h
25. Verify MBARA + Offset 20h[28] = ‘1b’, set MBARA + Offset 20h = ‘43F6460h’.
26. Wait 4ms.
27. Verify MBARA + Offset 20h[28] = ‘1b’, set MBARA + Offset 20h = ‘4310042h’.
28. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 43F6020h
29. Wait 4ms.
30. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 8310000h
31. Verify MBARA + 20h[28] = 1b, TEMP[15:0] = MBARA + 20h [15:0]
32. Verify MBARA + 20h[28] = 1b, set MBARA + 20h = 4310000h or with the TEMP[15:0] or with 10h
33. Verify MBARA + 20h[28] = 1b
34. Clear MBARA + F00h[5] = 0b (Read-Modify-Write).
35. Ensure SLP_LAN# is programmed properly and it is not overwritten by platform codes.

10.6 Additional GbE-based Wake Events

The LAN_WAKE# is used as a wake pin when the GbE PHY LANWAKE pin is connected on. The System BIOS should enable this as wake event by setting the LAN_WAKE# Enable bit (LAN_WAKE_EN ABASE + 9Ch[16]). Unlike other wake events the System BIOS does not need to clear the corresponding LAN_WAKE# Status bit (LAN_WAKE_STS ABASE + 8Ch[16]) as the bit will be cleared by the hardware. If Wake from DeepSx is required, LAN_WAKE# wake from DeepSx can be configured by setting PWRMBAR + 34h [0] to 1b.

10.7 Retrieve Lan Phy revision

Lan Connected Device varies in its revisions. BIOS handles gathering this information and displays it in Platform Information Menu. To get this data MAC and PHY have to establish communication through MDI commands:

1. Set PCI Gbe offset + Command register 04h = 07h to enable PCI Commands
2. Acquire Gbe Semaphore - ensure MBARA + F00h[5] = 1b
    a. Or MBARA + F00h[5] (set bit 5).
    b. Read MBARA + F00h.
    c. If MBARA + F00h [5] = 1b (true) continue
3. Write to MDI Control register MBARA + 20h value 43f6020h
4. Read back MBARA + MDI Control Register 20h[28] and verify that ready BIT 28 is set to 1b.
5. Repeat verification if not passed until given time of 40 000 microseconds. If timeout is reached there is a chance that MAC and PHY are not on the same bus. To fix it set MBARA + Extended Device Control Register 18h[11] to 1b which will cause MAC and PHY to be on SMBus. From this point proceed to step 3 and 4 which should now pass.
6. Wait 4 ms.
7. Set MBARA + 20h = 4302580h.
8. Verify MBARA + 20h[28] = 1b.
9. Set MBARA + 20h = 8430000h.
10. Verify MBARA + 20h[28] = 1b.
11. Read MBARA + 20h[7:0] – this value represents retrieved LAN Phy revision.
12. Set MBARA + 20h = 4302180h.
13. Verify MBARA + 20h[28] = 1b.
14. Release GBE Semaphore: MBARA + F00h[5] = 0b

10.8 Power Optimizer Configuration

BIOS must set Intel Gbe PCI offset 80h[5] = 1b to enable power gating.

10.8.1 LTR Programming

The LAN Controller reports the latency tolerance requirements for memory reads and writes to support entry to lower system power management states.

The System BIOS must program

- the maximum snoop/non-snoop platform latency values to 00000846h in the GbE controller's PCI LTR capability register at GbE PCI offset 0A8h.

In order to support assignment and reassignment of PCI IRQs, the OS will need to know how the system board has wired each motherboard integrated PCI device and each PCI slots interrupt pins to the PCI Interrupt Router's interrupt pins. This chapter describes the capabilities PCH has to allow customization of such routings, and how System BIOS should report this platform-specific information to the OS.

10.8.2 Clock and Lock Control Configuration

Lan Controller might block floor power of the PCH. In order to avoid this BIOS must program CLCTL at Gbe PCI offset 94h. Byte 2 and 3 are reserved. Byte 1 is responsible for Lock Control and Byte 0 for Clock Control. All of the settings are in bit 0 to bit 4. When Gbe is enabled all of those bits have to be set by the System BIOS to 1. Therefore BIOS needs to program Gbe PCI Config + 94h to 1Fh.

§ §

11 SMBus

PCH provides a System Management Bus (SMBus) 2.0 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). PCH is also capable of operating in a mode in which it can communicate with I^2C compatible devices.

11.1 SMBus Usage Model

The Host Busy bit located in the SMBus Host Status Register, SMBBASE + 00h[0], can clear before the correct data is loaded in the data register. The wrong data may be read from the SMBus controller's data register. The System BIOS should use the INTR bit, SMBBASE + 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, SMBBASE + 00h[0], should be zero indicating that the SMBus Host controller is not busy. Also if the Host Busy bit doesn't 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.

11.2 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 SMBBASE + 00h to 1Eh.
5. Program the SMBus Transmit Slave Address Register with the DIMM SMBus address, SMBBASE + 04h. For example: If the DIMM address is A2h and a byte is to be read from the DIMM, program SMBBASE + 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, SMBBASE + 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, SMBBASE + 02h[7:0] to 48h.
8. Wait on the Host Busy (HOST_BUSY) bit to be clear in the Host Status Register, SMBBASE + 00h[0]. The System BIOS should use the INTR bit, SMBBASE + 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 clear, check the DEV_ERR bit in the Host Status Register, SMBBASE + 00h[2] = 0, if the HOST_BUSY bit is clear and the DEV_ERR bit is clear the BIOS 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.

11.3 Block Mode

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

11.3.1 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.

11.3.2 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 last byte then end, otherwise loop to step 7.

11.4 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 loads. 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 System BIOS is done 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).

11.5 Wake Up from SMBus

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

11.6 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.

11.7 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 transfer are verified.
7. Clear the Byte Done Status bit at the end of the transfer.

11.8 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.

11.9 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. An 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.

11.10 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.

Register.Field	Value	Comments
SMBBASE + 11h[2]	X	BIOS must set this bit to the proper value to allow generation of SMI# or interrupts due to SMBALERT#
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.

11.11 SMI/OS Driver Resource Arbitration

System designs may request that the PCH support a usage model that allows both an OS present 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 either 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.

11.11.1 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 the responsibility of the System BIOS SMI functionality; at no time should the OS present 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 PCH provides a simple mechanism for software in two different execution contexts to arbitrate for ownership of the I/O resources of the PCH 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.

11.11.2 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.
2. SMI software decides it needs to run a SMBus transaction.
3. 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.
4. SMI software saves the state of the INTREN, SMBALERT_DIS, and HOST_NOTIFY_INTREN bits.
5. SMI software disables all SMBus interrupts by programming INTREN to 0, SMBALERT_DIS to 1, and HOST_NOTIFY_INTREN to 0.
6. SMI switches the SMBus interrupt to SMI by setting PCI offset 40h[1].
7. 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.

8. SMI may exit back to OS context and be re-entered when the transaction is completed with an SMI. Note: the INUSE_STS bit should still be set as the SMI still owns the SMBus host controller I/O space.
9. 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.
10. SMI switches the SMBus interrupt back to PCI-based interrupt by clearing PCI offset 40h[1].
11. SMI software restores the state of the INTREN, SMBALERT_DIS, and HOST_NOTIFY_INTREN bits to the values saved in step 4.
12. 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.

11.12 SMBus Handling for System Resets

During a SMBus transaction in which an I^2C device is sending information to the PCH, the I^2C device may not release the SMBus if the PCH 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:

1. 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.

2. 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 an SMBus clock operating at 3.2KHz or higher.
4. This workaround does not take ASF requirements into account.

11.13 Additional Programming Requirement

BIOS needs to program the follow register for PCH SMBus in BIOS early phase to ensure it operates under correct frequency.

1. Set SMBus PCI 64h = 0A0A0000h

§ §

12 Extensible Device Controller Interface (xDCI)

Skylake and Kaby Lake PCH USB supports device mode (xDCI), and the Host/Device role control is by using the ID pin. The ID pin is an input to the host controller from a USB Type-C connector and driving the Host/Device roles.

12.1 xDCI Enabling and Disabling

System BIOS should setup the following registers:

- Set PMCTL register PCR[OTG] + 1D0h[5:0] to 111111b
- Set PCR[OTG] + A2h [2] to 1b

To enable xDCI

1. Set DUAL_ROLE_CFG0, xHCI MBAR + 80D8h bit [21, 20] to [0, 0]
2. Set xDCI PCI offset A2h [2:1] to 1b, 0b

To disable xDCI

1. Set DUAL_ROLE_CFG0, xHCI MBAR + 80D8h bit [21] to 1b
2. Set DUAL_ROLE_CFG0, xHCI MBAR + 80D8h bit [20] to 0b
3. Wait for host owned (DUAL_ROLE_CFG, xHCI MBAR + 80DCh bit [29]) SS DRD mode to be 1.
4. Set xDCIBAR + C110h [1] to 1b
5. Set xDCI BAR + C200h [6] to 1b
6. Set xDCI BAR + C2C0h [27] to 1b
7. Put xDCI into core D3 (xDCI BAR + 10F818h [1:0] = 11b)
8. Wait for core D3 (xDCI BAR + 10F810h [11:8] = 1111b)
9. Set xDCI PCI offset A2h [2, 1] to [1, 0]
10. Set the PMCSR to D3 (xDCI PCI offset 84h bit [1:0] =11b)
11. Set PCR[PSF_2] + "AGNT_T0_SHDW_PCIEN"[XDCI][8] to 1
12. Set PWRMBSE + 628h [24] to 1b to disable xDCI Controlle in PMC

Note: DUAL_ROLE_CFG register is in the xHCI MMIO, instead of xDCI space, because this is an OTG function.

Note: If xHCI is disabled, the BIOS will also disable the xDCI

12.2 PCI/ACPI Modes

While the xDCI can be configured to work in either PCI or ACPI modes, the POR is to support PCI mode only. The configuration is done via a write to the IOSF2OCP private register by a sideband access.

12.2.1 PCI Mode of Operation

System BIOS can decide to leave xDCI to operate in PCI mode so that operating system can discover the device when doing PCI scan. BIOS will have to disable BAR1 by setting PCICFGCTRL.BAR1_DISABLE to 1b in xDCI private config space before PCI enumeration.

1. PCICFGCTRL.BAR1_DISABLE, offset 200h [7] = 1b.
2. Set PCR[PSF_2] + "AGNT_T0_SHDW_PCIEN"[XDCI][19:18] to 11b

12.2.2 PCI Mode Interrupt Pin Mapping

xDCI device interrupt pin (IPIN) could be mapped into one of the INTx (A/B/C/D) in PCI mode by manipulating "PCICFGCTRL.Interrupt Pin", PCR[OTG] + 200h bit [11:8] to allow maximum flexibility for interrupt routing. It's INTx can map to the IRQ direct by manipulating the "PCICFGCTRL.PCI_IRQ", PCR[OTG] + 200h bit [27:20]. Refer to section 6.1.2 for further details and restriction in assignment.

12.3 Power Management Requirement

1. Set xDCI PCI offset A2h [2:1] to 10b (purpose is to enable D3HEN)
2. Set xDCIBAR + C200h [6] to 1b (enable the USB 2.0 PHY suspend just in case if xDCI driver is not loaded.)
3. Set "PMCTL", PCR[OTG] + 1D0h [5: 0] to 111111b

§ §

13 XHCI Support

The Skylake/Kaby Lake PCH implements xHCI host controllers (D20:F0) which support up to up to 10 USB 3.0 ports and 13 USB 2.0 ports, one port reserved for USBr.

The Skylake/Kaby Lake PCH-LP implements xHCI host controller (D20:F0) which supports up to 6 USB 3.0 ports and 11 USB 2.0 ports, one port reserved for USBr.

On certain USB 2.0/3.0 ports, they can be further configured to support device mode (dual role). Dual role port is using micro-AB receptacle. Micro-A plug results in host mode, while micro-B plug results in device/peripheral mode.

On the USB 3.0 ports, they can also be configured to support SSIC.

13.1 BIOS Core Requirements

System BIOS is required to have native pre-boot xHCI driver support and capability to support port 60h/64h emulation through the XHCI interface.

It is expected BIOS recovery to be supported over XHCI interface. Thus, all necessary Intel XHC specific initialization has to be done before it, specifically, in PEI.

13.2 Legacy USB Support

The following sections provide information regarding Legacy USB Support.

13.2.1 Legacy USB Support and Dual Processor Issues

In dual processor platforms, PCH does not provide a mechanism to determine which processor is responsible for generating a port 60/64 SMI trap, so it is not possible to determine which processor should receive the I/O restart. Setting the I/O restart on a processor that did not generate the SMI will result in unpredictable behavior. The System BIOS must use other mechanisms to determine the source of the trap or the System BIOS should not use the trap for the legacy emulation.

To determine the processor that caused the trap, the System BIOS must disassemble the instruction stream to find the I/O instruction that caused the I/O trap. Some of the issues that must be addressed with this method include protected mode, page tables, and virtual x86 mode disassembly. An assumption must also be made that the instruction that caused the SMI trap was either an immediate I/O through al, like “out 64h,al”, or by an indirect access such as “in dx,al” when dx = 60 or 64h. There exists an obscure case where the reps instruction can generate the SMI. In such a case, the reverse disassembly of the instruction stream is not possible. However, this case is not known to ever occur in today’s systems and is not likely to occur in future systems.

An alternative method is for the legacy SMI handler not to enable the legacy KB SMI traps in DP* mode. In this case, the SMI generated at the end of the USB schedule loop can be used to poll the current state of the numlock, caps and scroll LEDs, which can then be copied to the USB keyboard. Data read from the USB keyboard can be

written back into the KBC (using the D2 command), emulating the keypress. This method requires the physical connection of an 8042 compatible KBC.

13.2.2 Handoff of USB Controller to the OS

The OS driver will go through the following steps for taking control of the host controller from the System BIOS. There is no specific handshake mechanism to inform the System BIOS that the OS is taking over.

13.2.3 Implementing Legacy USB Keyboard Support

In today's PC environment, windows-based application software typically does not directly access I/O locations for the keyboard and mouse functions. The system software provides this function and the interface is comprehended by the OS. However, in earlier environments some application software directly accessed the keyboard controller's I/O space. In most desktop PC systems the keyboard controller is based on the 8042 controller.

For a USB-based keyboard and mouse to be software compatible with an 8042 controller-based keyboard and mouse, older software must be able to access the same I/O locations with the same values. Furthermore, the time delays associated with the ports must be roughly the same.

The 8042-based keyboard controller (KBC) typically has the following characteristics:

- It provides serial interfaces for both the keyboard and PS/2-style mouse.
- The 8042 controller's registers are mapped to the ISA I/O addresses 60h and 64h.
- The 8042 controller asserts two interrupt lines; IRQ1 for the keyboard and IRQ12 for the mouse.

This section provides a legacy PCI device implementation example that supports the 8042 controller-based keyboard and mouse commands in a USB environment. The legacy software support method described requires a combination of hardware and software.

8042 controller emulation is only needed in certain OS environments when applications are permitted to directly access the I/O space.

Note: This implementation is not the only possible solution for providing this legacy support.

13.2.4 USB Legacy Operation Requirements

Software that is run early in the boot process must be modified to configure the USB controller, enable the keyboard and mouse, and set up the USB controller's scheduler.

The major assumptions are:

- A standard 8042 device-like keyboard controller is present in the system at I/O locations 60h and 64h. The control logic described here must be able to selectively intercept or pass through I/O cycles targeted for the KBC.
- No changes are needed to the 8042 keyboard controller code.

- USB aware operating systems handle the host controller completely without any legacy hardware or software support. No operating system should take control of the USB host controller while the Legacy Support hardware and software are enabled.

For information on the USB Legacy Keyboard/Mouse Control Register, see the PCH EDS.

13.2.4.1 Commands Going to the KBC, Keyboard, Or Mouse

Some legacy software will attempt to send commands directly to the keyboard controller (8042 device). In addition, some legacy software may also send commands directly to the keyboard or mouse, although this is rare. This case is typically done by the System BIOS, which would be already aware of the peripheral being on USB. The keyboard and mouse commands perform operations like resetting the keyboard, setting the typematic rate, turning on or off LEDs, or changing the scan-code translation method. The software writes the command into the 8042 keyboard controller. The 8042 keyboard controller sends the command out serially to the keyboard or mouse. An interrupt can be generated when the action completes. A USB-based keyboard has similar commands (reset, turn on/off LEDs, etc.) and the commands are sent via USB.

This leads to the following requirements:

- Since no changes are desired in the 8042 controller code, the combination of software and system logic must determine if the data going to the 8042 controller should be diverted to the USB keyboard. The logic must be able to generate an SMI# based on command writes to the keyboard controller. Furthermore, for multi-byte commands, the software must be able to generate an SMI# during a read cycle (which might occur in-between subsequent writes).
- The logic must be able to selectively generate an SMI# based on reads or writes to the keyboard controller. Cycles where an SMI# is generated are not passed to the keyboard controller.
- There must be SMM code to interpret the commands and status results associated with the keyboard controller.
    - If an SMI is generated from legacy keyboard/mouse support from Device 26 controller, software or BIOS might not be able to clear the appropriate status bits in Device 26. However BIOS can clear the relevant SMI status from Device 29 instead. That means BIOS has to enable at least one USB controller in Device 29 in order to provide legacy USB support.
- For one specific case, the keyboard controller is used to control the A20GATE signal. In this case, it is not desired to invoke an SMI# due to the great performance degradation. To handle this specific case, the logic must “pass” commands associated with setting and resetting the A20GATE signal. Thus, it is required that the logic must pass A20GATE sequences to the external 8042 controller. This is called the “A20GATE Pass Through Sequence”.

13.2.4.2 Keystroke or Mouse Data Received

In legacy hardware implementations, one of the more common actions performed by the 8042 controller is receiving keystroke or mouse movement data. The keyboard controller de-serializes the incoming data streams, generates an interrupt, provides a

status value in the status register, and finally provides the data bytes associated with the keystroke or mouse movement.

A USB-based keyboard or mouse will be similar. However, the USB controller will store the incoming data in main memory, rather than at an I/O location. This method requires that SMM code be invoked based on the incoming USB data.

The SMM code manages the USB data queues, takes the received data, and writes it into the 8042 device using the "D2" (keyboard) or "D3" (mouse) commands. This instructs the 8042 controller to process the incoming data as if it had come from the standard PS/2 keyboard or mouse.

Once the data is placed in its registers, the 8042 controller generates the appropriate interrupt (IRQ1 for the keyboard or IRQ12 for the mouse). The normal interrupt handlers then process the data.

To take advantage of this scheme, the SMM code must be invoked (via SMI#). The USB controller must also be able to generate an SMI# based on the data being stored to main memory. This results in the following requirements:

- There must be the ability to generate an SMI# based on a USB interrupt.
- There must be additional code to move the data to the appropriate registers and manage the USB controller during runtime. It is envisioned that this would be part of the SMM code.

13.3 Software IRQ Generation in Platform Controller Hub (PCH)

The trapping of Legacy KB/MS has been changed; please refer to new register added in EDS spec update for more details. The register can be used is called ULKMC and PCH no longer generates regular interrupt from the controller, only SMI. The following section described how that works in general using the controller.

PCH has the capability to generate an IRQ1 or IRQ12 by using the LEGACY_ELIM Register, TCOBASE + 10h. This can generate IRQ's in the absence of a KBC in the system. See the PCH EDS for a description of this register.

The IRQ1 and IRQ12 sources are each logically AND'ed with the respective IRQ1 and IRQ12 that come from the SERIRQ logic. This is necessary because the SERIRQ logic will report IRQ1 and IRQ12 to be high (active), since there is no Super I/O to drive them low.

Note: In a system that does have a Super I/O, the new bits must be left at 1, since it is AND'ed with the Super I/O. Don't attempt to write these bits to 0 in a system that has a KBC; it will not be validated, and is highly likely to break software.

The following algorithm assumes the byte is being sent from the keyboard. The byte being sent from the mouse is equivalent.

1. An SMI is received and discovered to be a USB interrupt.
2. The interrupt is discovered to be due to a Transfer Descriptor associated with a keyboard device.

3. The data is analyzed and it is determined that the interrupt is due to a new key press. The USB key-code is translated into the equivalent scan code set 2 (SS2) PS/2 scan code.
4. The result is queued on a queue of data to be sent from the keyboard to the system.
5. Other USB interrupts are handled.
6. The keyboard controller emulation code is executed at exit. It determines if the conditions are correct to return a byte to the system (for example, emulated OBF indicates empty, keyboard interface not disabled, etc.). If a byte isn't returned, then the emulation exits awaiting the next event.
7. The queue of data to be sent from the keyboard to the system is found to contain a byte to be returned.
8. Given the typical keyboard controller configuration, it is translated from SS2 to SS1.
9. The byte to be returned is stored in the emulated output buffer.
10. The emulation does the out to the port to enable IRQ 1.
11. The emulation exits.
12. Time passes and the processor continues execution.
13. The system code services the interrupt and reads port 60h.
14. The controller traps the read and causes an SMI trap.
15. The trap is determined to be caused by the read from port 60h (TRAPBY60R in USB_LEGKEY).
16. The emulation code clears the interrupt register thus turning off IRQ 1 and / or IRQ 12. Note: The emulation code can validly do this each time there is a read from 60h since that is what the keyboard controller would do as well. (There is no time where both IRQ 1 and IRQ 12 should be asserted simultaneously by either the KBC or the emulated KBC. This would be a violation of the KBC / system protocol.)

13.4 xHCI Controller Initialization

The PCH Init driver 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.
- BIOS initializes xHCI registers settings
- PCI enumeration performed on xHCI controller.

If BIOS need to disable XHCI compliance test mode, it must be done after XHCI memory space assigned and before XHCI Host Controller initialization by set xHCI MBAR + 80ECh [0] = 1b

Compliance Mode is recommended to be default enabled. For USB Type-C support, enable Compliance Mode for testing.

13.4.1 xHCI Controller Setup

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

1. Program xHCI PCI offset 40h[21, 20, 18, 8] to 1b, 1b, 1b, 1b
2. Program xHCI PCI offset 44h[19, 18, 17, 16, 15, 14, 10, 9, 7, 3] to 1b, 1b, 1b, 1b, 1b, 1b, 1b, 1b, 1b, 1b
3. Program xHCI PCI offset 50h = 0FCE6E5Fh
4. Set xHCI MBAR + 80C0h[10] to 0b (This bit must be set to “1b” when platform is going to enter S3 state)
5. Program xHCI PCI offset 74h[1,0] = 1b, 1b

Delay 20us.

6. Program xHCI PCI offset 74h[1,0] = 0b, 0b
7. Program xHCI PCI offset 40h[21, 20, 18, 8] to 1b, 1b, 1b, 1b
8. Program xHCI PCI offset 44h[19, 18, 17, 16, 15, 14, 10, 9, 7, 3] to 1b, 1b, 1b, 1b, 1b, 1b, 1b, 1b, 1b, 1b
9. Program xHCI PCI offset 50h = 0FCE6E5Fh
10. Program xHCI PCI offset A2h[3, 2] to 0b, 1b
11. Program xHCI PCI offset ACh[18] to 1b
12. Program xHCI PCI offset A4h[13, 12, 11] to 0b, 1b, 1b
13. Program xHCI PCI offset A8h[17, 14] to 1b, 1b
14. Set xHCI MBAR + 0Ch [31:16] = 200h, [7:0] = 0Ah
15. Set xHCI MBAR + 8008h[19] to 0b
16. Set xHCI MBAR + 80A4h[30, 27, 24, 23, 16, 7, 4, 3, 2] to 1b, 1b, 1b, 1b, 0b, 1b, 1b, 1b, 1b
17. Set xHCI MBAR + 80A8h[24] to 1b
18. Set xHCI MBAR + 80E0h[22, 16, 9, 6] to 0b, 0b, 0b, 1b
19. Set xHCI MBAR + 80F0h[20] to 0b
20. Set xHCI MBAR + 80FCh[25] to 1b
21. Set xHCI MBAR + 8154h[21,13] to 0b, 1b
22. Set xHCI MBAR + 816Ch[19, 18, 17, 14, 13, 12, 11, 10, 9, 8, 4, 3, 2] to 1b, 1b, 1b, 1b, 0b, 0b, 0b, 0b, 0b, 0b, 1b, 1b, 1b
23. Set xHCI MBAR + 8140h = hFF08003Ch for SPT-LP A/B and SPT-H A/B/C steppings. Set xHCI MBAR + 8140h = FF00F03Ch for all others steppings
24. Set xHCI MBAR + 81C4h [11, 8, 3 ,2, 1, 0] to 1b, 1b, 0b, 0b, 1b, 1b
25. Set xHCI MBAR + 8094h[23, 22, 6] to 1b, 0b, 1b
26. Set xHCI MBAR + 809Ch[31] to 0b
27. Set xHCI MBAR + 80B0h [24] 0b, [23] 1b, [18:16] 000b
28. Set xHCI MBAR + 80B4h[5, 2] to 1b, 1b - only for SPT-LP A/B step and SPT-H A/B/C steppings
29. Set xHCI MBAR + 80B8h[30] to 1b
30. Set xHCI MBAR + 80ECh[19, 17] to 1b, 1b
31. Set xHCI MBAR + 8110h[20,11,2] to 1b, 1b, 0b
32. Set xHCI MBAR + 8144h[8] to 1b
33. Set xHCI MBAR + 8164h[7, 6, 5, 4, 3, 2] to 1b, 1b, 1b, 1b, 1b, 1b
34. Set xHCI MBAR + 8174h[24] to 1b, [11:0] = C0Ah

35. Set xHCI MBAR + 817Ch[28:16] 0332h, [12:0] 00A3h
36. Set xHCI MBAR + 8180h[28:16] to 00CBh, [12:0] 0028h
37. Set xHCI MBAR + 8184h[28:16] to 0064h, [12:0] 001Eh
38. Set xHCI MBAR + 818Ch[16, 8] to 1b, 0b
39. Set xHCI MBAR + 8194h[25] to 1b
40. Set xHCI MBAR + 819Ch[20] 1b, [16] 1b, [14] 1b, [13] 1b, [12] 1b, [11:8] 3h, [7:4] 7h, [3:0] Dh
41. Set xHCI MBAR + 81B4h[9:0] to 0h
42. Set xHCI MBAR + 81BCh[31] to 1b
43. Set xHCI MBAR + 8758h[0] to 1b

BIOS must perform USB 2.0 high-speed signals quality tuning by perform below steps through sideband interface access.

1. Set PCR[USB2] + 402Bh[22] = 1b
2. Set PCR[USB2] + 4001h[25] = 1b
3. Set PCR[USB2] + 7034h[31:0] = 50000000h
4. Set PCR[USB2] + 7038h[31:0] = 0A70FCF6h
5. Set PCR[USB2] + 703Ch[31:0] = 1FBBC000h
6. Set PCR[USB2] + 7F04h[10:9] = 11b

13.4.2 Super Speed Port Enable

"USB3 Port Enable" bits in SSPE register (Address offset: xHCI MBAR + 80B8h) can be set to 1b only when USB3PDO bits in "USB3 Port Disable Override" register (Address offset: xHCI MBAR + 84FCh) are set to 0b.

13.4.3 xHCI SSIC Interface Configuration

System BIOS must execute the following steps as part of System BIOS initialization of the PCH xHCI SSIC Configuration:

1. If XHCI MBAR + 8420h[2:1] = 00b, it means XHCI SSIC disabled by straps, then perform below 2 settings.
   - Set XHCI MBAR + 880Ch[31:30] = 11b
   - Set XHCI MBAR + 883Ch[31:30] = 11b
2. Set PCR[MMP0] + F00Ch[5:0] to 33h,
3. Set PCR[MMP0] + F038h[0] to 1b
4. Lane 0 settings:
   - Set PCR[MMP0] + 1050h [24] = 1b, [7:0] = 20h
   - Set cnfgupd bit PCR[MMP0] + 1054h [26] to 1b to update effectively configuration registers from shadow registers.

- Set PCR[MMP0] + 1058h[16] to 1b
- Set cnfgupd bit PCR[MMP0] + 1054h[26] to 1 to update effectively configuration registers from shadow registers.
- Set PCR[MMP0] + 1060h[23:16] to 0h
- Set cnfgupd bit PCR[MMP0] + 1054h [26] to 1b to update effectively configuration registers from shadow registers.

5. Lane 1 settings
   - Set PCR[MMP0] + 3050h [24] = 1b, [7:0] = 20h
   - Set cnfgupd bit PCR[MMP0] + 3054h[26] to 1 to update effectively configuration registers from shadow registers.
   - Set PCR[MMP0] + 3058h [16] to 1b
   - Set cnfgupd bit PCR[MMP0] + 3054h[26] to 1 to update effectively configuration registers from shadow registers.
   - Set PCR[MMP0] + 3060h[23:16] to 0h
   - Set cnfgupd bit PCR[MMP0] + 3054h[26] to 1 to update effectively configuration registers from shadow registers.
   - Set PCR[MMP0] + F03Ch[16,8] to 1b, 1b
   - Set PCR[MMP0] + F03Ch[0] to 1b to update effectively configuration registers from shadow registers.
   - Set xHCI MBAR + 8804h[3] to 0b, [17:16] to 01b
6. Profile Attributes for Port 1
   - Set xHCI MBAR + 816Ch[5] to 1b
   - SSIC Series:

     Set xHCI MBAR + 8808h[4] = 0b (Series B)
   - SSIC Gear:

     Set xHCI MBAR + 8808h[3:2] = 0b (Gear 1)

     Issue RRAP cmd to local PHY:

     0022C002h

     00A2C002h

   Issue RRAP cmd to remote PHY:

     00228002h

     00A28002h
   - SSIC Disable Data Scrambling:

     Set xHCI MBAR + 880Ch[25] to 1b

     Issue RRAP cmd to remote PHY:

     04038001h

- Set xHCI MBAR + 8810h[2] to 1b

  Issue RRAP cmd to remote PHY:

  04048001h

7. Profile Attributes: Port 2
   - Set xHCI MBAR + 816Ch[5] to 1b
   - SSIC Series:

     Set xHCI MBAR + 8838h[4] to 0b (Series B)
   - SSIC Gear:

     Set xHCI MBAR + 8838h[3:2] to 0b (Gear 1)

     Issue RRAP cmd to local PHY:

     0022C002h

     00A2C002h

     Issue RRAP cmd to remote PHY:

     00228002h

     00A28002h

     Issue RRAP cmd to remote PHY:

     0033C00Ah
   - SSIC Disable Data Scrambling

     Set xHCI MBAR + 883Ch[25] to 1b

     Issue RRAP cmd:

     04038001h
   - SSIC U0 Stall

     Set xHCI MBAR + 8840h[2] to 1h

     Issue RRAP cmd to remote PHY:

     04048001h
8. Issue an RRAP commands for port 1

   Set xHCI MBAR + 8904h [25] to 1h, [23] to 1h
9. Issue an RRAP commands for port 2

   Set xHCI MBAR + 8A14h [25] to 1h, [23] to 1h

   Ports enabled
10. Set xHCI MBAR + 880Ch [24:21] to 0h

11. Enable Port 1 configuration

    Set xHCI MBAR + 880Ch

        bit [30] to 1h,

        bit [31] to 1h if the port has no physical SSIC Device connected

12. Enable Port 2 configuration

    Set xHCI MBAR + 883Ch

        bit [30] to 1h,

        bit [31] to 1h if the port has no physical SSIC Device connected

13.4.4 Port Disable Override

In Skylake/Kaby Lake PCH, Port disable override on xHCI ports is implemented as extended capability, with id=C6h.

There is a corresponding disable bit for USB 2.0 ports and USB 3.0 ports (MBAR + offset 84F8h – USB2PDO and MBAR offset 84FCh – USB3PDO respectively). BIOS needs to program them accordingly. Since the registers are write-once and maintained by SUS Well, after BIOS program the bits, BIOS must read the value back and check whether the value was actually programmed properly; if not, BIOS need to perform a warm rest(by write 6h to IO port CF9h) to unlock these write-once registgers.

If the USB3PDO bit is to be set for unused port, BIOS must not ever set the corresponding SSPE bit in xHCI Memory Mapped register SSPE_REG – offset 80B8h Bits 9:0. When USB3.0 Hosting Direct Connect Interface (DCI) is enabled, BIOS should not set any of the USB3PDO bits.

13.4.5 Hiding/Disabling xHCI Controller

xHCI controller could be fused disabled (SKU dependent). Refer to section 5.3.1 on its handling.

It is not desired to disable the xHC because there are other device under the same device 20, such as xDC, Thermal and CIO2. Since the thermal is to be always enabled, disabling xHC which is function 0 will cause issue to those non-function 0 device under the same device number. If xHC is not used, we can rely on the fact that it will be put into D3 by the OS driver.

13.4.6 Locking xHCI Register Settings

After xHCI is initialized, BIOS should lock the xHCI configuration registers to RO. This prevents any unintended changes. There is also a lockdown feature for OverCurrent registers. BIOS should set these bits to lock down the settings prior to end of POST.

1. Set Access Control bit at xHCI PCI offset 40h[31] to ‘1b’ to lock xHCI register settings.
2. Set OC Configuration Done bit at xHCI PCI offset 44h[31] to ‘1b’ to lock overcurrent mappings from further changes.

13.4.7 xHCI Legacy SMI Enabling

In order to enable xHCI Legacy SMI[4], take the following steps:

1. Set ABASE + 30h[31] = 1b
2. Set “USB SMI Enable” in “USB Legacy Support Capability” structure to 1

If the SMI is triggered and LPC PCI register offset 94h[11:8] is set, BIOS must clear the current SMI status by write “1” to LPC PCI register offset 94h[11:8] before doing the 2nd 60/64 access. This should be done by legacy USB driver.

13.4.8 Host Controller Reset or WPR

Skylake PCH introduces WPR debug feature which, when enabled, leads to USB3.0 Rx terminations to be enabled earlier than we normally would. This may lead to USB3.0 devices to enter Compliance mode – which is not a problem if BIOS performs HC Reset on G3 exits as HC Reset would retrain the devices.

HCReset would have impact on DbC enabling, thus it would be better to perform WPR on all SS ports that are without DbC enabled.

13.4.8.1 Additional Recommendation of Host Controller Reset

It is recommended to add 1ms short delay to avoid race condition after write USBCMD.HCRST for issuing HC reset and before accessing any MMIO register, including reading USBCMD for polling HC reset process completed. There is only 1ms impact to BIOS POST time since the reset command only be executed once during POST. Below is example for BIOS implementation. Please refer to the appropriate PCH EDS for USBCMD register detailed information.

```
XhcSetOpRegBit (Xhc, XHC_USBCMD_OFFSET, XHC_USBCMD_RESET);

gBS->Stall (XHC_POLL_DELAY);      // XHC_POLL_DELAY is 1000 as 1 ms delay

Status = XhcWaitOpRegBit (Xhc, XHC_USBCMD_OFFSET, XHC_USBCMD_RESET,
FALSE, Timeout);
```

Without this implementation in BIOS, customer may meet sighting#4772339 that was documented in Doc#552319 - Skylake Sighting Report. Sighting#4772339 – Possible Global Reset or Processor Catastrophic Error During Reboots/Sx Cycling.

13.5 D3 enabling

xHCI support D3-hot in ACPI _S0W method only when Intel USB filter driver installed and _DSM method is evaluated by driver. Please be noticed that USB filter driver (v1.0.0.37 and later version) is required to support this BIOS change.

[4] The interrupt status bits of D31:F0:Reg 94h[11:8] must be clear by writing all 1’s before next SMI by IO 0x60/0x64

13.6 Debug Capabilities Support

The XHC implements a Debug Capability (DbC) as defined by the USB Debug Class Specification and the XHCI spec.

The DbC implements two types of interfaces:

1. Kernel Debug:One General Purpose (DbC.GP1) interface as defined by the USB Debug Class Specification

    a. This interface is controlled by the XHCI spec defined Debug Device interface specification.

2. Platform Debug: Three interfaces (details below) which are all HW controlled interfaces used exclusively for Platform debug and DnX use models.

    b. One General Purpose (DbC.GP2) interface (as defined by the USB Debug Class Spec.) that is used for moving data to and from any memory mapped address (from various predefined address spaces) to the external debug host.

    c. One DFx (DbC.DFX) interface (as defined by the USB Debug Class Spec.) that is used by the Debug Host to communicate with the DCI bridge.

    d. One Trace (DbC.Trace) interface (as defined by the USB Debug Class Spec.) that is used by the Debug Host to extract platform (HW or SW) trace data provided by the Intel® Trace Hub infrastructure within the SOC or PCH/processor combination.

For the Skylake/Kaby Lake PCH implementation, at any given time, only one DbC interface can be used. The DbC mode that can be supported are DCI mode, Kernel Debug Mode.

DCI mode:

- Default Si without any register programming
- can be exited through
    - Target running kernel debug mode
    - Physical disconnect and reconnect happen (not sure how long the target and external host will wait for such connections …)
- BIOS debug dump would not work under this mode (require the same physical disconnect and reconnect)

Kernel Debug Mode (BIOS default):

- with the “Force Disconnect on DCE” bit, XhcBar + 8740h + 18h bit[0] set
- allow switch over to kernel debug mode on DCE setting
- if the said BIOS (debug build) is configure to do debug dump through XHC, the switch over to Kernel debug mode will happen within the BIOS post.
- If the Intel TH policy is to use the DCI mode for trace, this mode will break it (when DCE is set by the target kernel). This mode will have higher priority than the Intel TH policy, still allow Intel TH usage if DCE is not set.

13.7 DCI Support

DCI enables DFx access flexibility which provides closed chassis access to DFx hooks for silicon, platform and software debug. To enable this after manufacturing, its enable bit need to be set. It is expected that BIOS will provide policy to enable or disable DCI, and user would be able to use BIOS option to change this policy. The user changing the setting from disable to enable, is taken as a consent from the user to enable this DCI feature. The BIOS option is expected to highlight the consent.

In system supporting SLP S0 state and ModPHY Sus power gating feature, Suswell in ModPHY may be power gated when SLP S0 state is entered. This will cause DCI to fail. Due to this reason, for system debug using DCI, ModPHY sus power gate feature must be disabled.

Policy	BIOS Action
Enable	1. Disable SLP S0 and ModPHY sus power gating feature 2. Set HSWPGCR1, PWRMBASE offset 5D0h to 80000000h. This step has higher priority than the TraceHub mode disable HSWPGCR1 step in section 30.1 3. Set PCR[DCI] + 4h bit[4] to 1 4. Set PCR[DCI] + 4h bit[0] to 1 **NOTE:** Once bit [0] has been set to a ‘1, bit [4] may still be written to a ‘0. If that occurs, bit [4] will be able to be changed back to a ‘1. In other words, even while locked, it is still possible to disable DCI in the future. Note that bit [4] is maintained across platform reset, so once enabled, DCI will remain enabled until disabled by writing bit [4] to a ‘0 (regardless of the setting of the lock bit.)
Disable	1. Set PCR[DCI] + 4h bit[4] to 0 2. Set PCR[DCI] + 4h bit[0] to 1 **NOTE:** Once bit [0](Host DCI Lock) has been set to a ‘1, bit [4](Host DCI Enable) 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.

§ §

14 Integrated SATA Controllers

The Platform Controller Hub (PCH) Serial ATA (SATA) controller supports a maximum of 8 SATA ports for Skylake/Kaby Lake PCH-H and maximum of 3 SATA ports for Skylake/Kaby Lake PCH-LP. See EDS for details on number of ports and SATA speed support per SKU.

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, System BIOS is responsible for restoring all registers that it initialized during POST.

Intel® RST for PCIe* Storage Remapping feature has dependency on SATA initialization, particularly the "Ports Implemented" register initialization. It is best to have most SATA initialization done before the remapping initialization.

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 or no lanes assigned to SATA when in AHCI mode, perform disabling steps as indicated in section 14.3 and exit here
2. Perform the programming requirement steps as indicated in section 14.1.6.
3. Purely based on the policy, check if Intel® RST for PCIe* Storage Remapping feature is enabled and program the ASSEL and MSS in SATAGC (SATA PCI offset 9Ch, bit[2:0] and bit[4:3]) accordingly
   a. If remapping feature is enabled, program ASSEL to 110b, and MSS to 00b.
   b. Else, program ASSEL to 000b and MSS to 10b
4. Program PCS "Port X Enabled" (For Skylake/Kaby Lake PCH-LP, it is located at PCI offset 92h[2:0]. For Skylake/Kaby Lake PCH-H, it is located at PCI offset 94h[7:0]) to all 1b
5. Perform steps as indicated in section 14.1.3 to initialize the registers in ABAR.
6. Perform steps as indicated in section 14.2 for DevSleep enabling.
7. Perform the power optimizing steps as indicated in section 14.1.5.
8. Check SATA Controller mode, and program the SATA Mode Select as indicated in section 14.1.1.
   a. If RAID mode is detected, perform the steps as indicated in section 14.1.4.
9. 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. Hot Plug is enabled
c. SATA test mode is enabled
d. Port is configured as eSATA port

10. Set MAP “SATA PortX Disable” ” (For Skylake/Kaby Lake PCH-LP, it is located at PCI offset 90h[10:8]. For Skylake/Kaby Lake PCH-H, it is located at PCI offset 90h[23:16]) to 1b if the particular SATA port is disabled.
11. Program PCS “Port X Enabled” (For Skylake/Kaby Lake PCH-LP, it is located at PCI offset 92h[2:0]. For Skylake/Kaby Lake PCH-H, it is located at PCI offset 94h[7:0]) to 1b if the particular SATA port is enabled.
12. Program SATAGC, SATA PCI offset 9Ch bit[31] to 1b.

14.1.1 Setting the SATA Controller Mode

The System BIOS 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 (For Skylake/Kaby Lake PCH-LP, it is located in Port Mapping register, PCI offset 90h[6]; For Skylake/Kaby Lake PCH-H, it is located in SATA General Configuration register, PCI offset 9Ch[16]). The System BIOS must never change the SATA controller mode during runtime. Please note that the availability of the following modes is dependent on the SKU of PCH in use.

14.1.1.1 AHCI Mode

AHCI mode is selected by programming the SMS field (refer 14.1.1 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, D23: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 RAID Mode

RAID mode is selected by programming the SMS field (refer 14.1.1 above) to 1b. 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, D23:F0. In RAID mode, the Sub Class Code, PCI offset 0Ah, will be set to 04h. This mode does require specific OS driver support.

In order for the RAID Option ROM to access all SATA ports, the RAID Option ROM enables and uses the AHCI programming interface by setting the AE bit, ABAR + 04h[31]. One consequence is that all register settings applicable to AHCI mode set by the BIOS have to be set in RAID as well. The other consequence is that the BIOS are required to provide AHCI support to ATAPI SATA devices which the RAID Option ROM does not handle.

PCH supports stable image compatible ID. When the alternative ID enable, PCI offset 9Ch [7] is 1b, PCH SATA controller will report Device ID as 2822h (PCH-H) / 282Ah (PCH-LP) / 2826h (if PCI offset 9Ch [6] is set for RSTe). Both PCI offset 9Ch [7] and PCI offset 9Ch [6], and same register bit PCI offset 9Ch [5] and PCI offset 9Ch [2:0] need to be programmed in single configuration write cycle as there are RWO bits. Refer to section 24.3 step 3 for PCI offset 9Ch [2:0] programming needs.

14.1.1.3 SATA Mode Setup Option

The System BIOS 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 92h. It is important to note that from Skylake PCH onwards, 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 and RAID modes, the System BIOS must program SATA PCI offset 92h[n:0] accordingly to ports muxing. If all physical layers (HSIO) are assigned to PCIe* lanes instead of SATA ports, or none of the ports are to be enabled because of no device connected and no ports are setup for hot plug or eSATA, then BIOS should have the SATA controller function disabled, refer to section 5.3 for details. However, if Intel® RST for PCIe* Storage is supported and the PCIe* storage device is detected, the SATA controller must be left enabled. 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, BIOS 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. See 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 System BIOS should reinitiate the detection sequence. If the device fails to respond within an additional ten seconds, the System BIOS should reinitiate the detection sequence again. If the device fails to respond within ten more seconds, the System BIOS can assume the device is not functioning properly.

It is recommended that Software or System BIOS 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 RAID or AHCI mode, the System BIOS must initialize the following memory-mapped AHCI registers specified by ABAR,PCI offset 24h:

- Program ABAR Size Select in SATA General Configuration register, PCI offset 9Ch[2:0] to value determined in 24.3 step 1-3, , then initialize the registers (CAP, PI, PxCMD) as indicated below:
- CAP register, Host Capabilities register (ABAR + 00h).
    - Set SIS (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 SIS (ABAR + 00h[26]) if SALP is supported to enable Aggressive Link Power Management (LPM) support
    - Set SIS (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
    - Clear EMS (ABAR + 00h[6]) if remapping is enabled; this is for disable Enclosure Management capability.
- 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 + 0Ch).

- Set PI (ABAR + 0Ch) to enable implemented SATA ports by setting respective PI bits to 1b.

Note: After BIOS 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.

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

- Set SUD, bit[1] to 1b if the SATA Controller supports staggered spin-up
- Set HPCP, bit[18] to 1b if Hot Plug is enabled for the particular port
- Set MPSP, bit[19] to 1b if Hot Plug is enabled and Mechanical Switch is attached for the particular port

- Set ESP, bit[21] to 1b if External SATA is supported for the particular port
- Set ALPE, bit[26] to 1b

Some of the bits in these registers are platform specific and must be programmed in accordance with the requirements of the platform. The details regarding how these registers must 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 System BIOS programs each bit at least once, even if the default setting of the bit is the desired value (i.e. System BIOS 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. Please refer to the PCH EDS to determine which bits are R/WO.

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 RAID Mode Initialization

When the SATA controller is initialized in RAID mode, the System BIOS must initialize the SATA controller and attached SATA devices as stated in the previous sections with the following exceptions:

- The System BIOS does NOT need to initialize DMA mode for HDDs discovered behind the SATA controller when in RAID mode. The RAID Option ROM will provide the initialization.
- SATA hard disk drives (HDDs) discovered by the System BIOS behind the SATA controller in RAID mode must NOT be added to the hard drive count at 40h:75h. The RAID option ROM will enumerate those drives and update the 40h:75h value accordingly. Updating the drive count at 40h:75h by BIOS will make it appear that more drives are attached to the system than are actually available.
- System BIOS must NOT install INT 13h support for the SATA HDD devices discovered and must NOT treat such devices as BAID (BIOS Aware Initial Program Load) devices, as the RAID option ROM implements the necessary INT 13h support for them.
- ATAPI devices attached to the SATA controller must be under the full control of the System BIOS and treated as BAID as they are in non-RAID mode.
- The System BIOS must load the RAID Option ROM when the controller's SCC (PCI offset 0Ah) returns 04h and VenderID/DeviceID match that of PCHR RAID SKU (refer to the PCH EDS or EDS spec update for DID/VID info).

14.1.4.1 RAID Option ROM

The Intel® Rapid Storage Technology RAID option ROM is a PnP option ROM that adds the Int13h services and provides a pre-OS user interface for the RAID implementation. The option ROM's Int13h services are used up to the point where the PCH RAID OS driver takes over.

The RAID option ROM is delivered as a single uncompressed binary image compiled for the 16-bit real mode environment. To conserve system flash space, the integrator may compress the image for inclusion into the System BIOS. System memory is taken from conventional DOS memory, attached to the EBDA, and is not returned.

The RAID option ROM will first attempt to rely on the interrupt-driven mechanism handled by System BIOS, and then will fall back to the polling mechanism if the System BIOS does not support the interrupt-driven mechanism. The determination is made via a memory semaphore at location 40:8E in BIOS Data Area (BDA).

Initialization:

- System BIOS configures the SATA controller for RAID mode, initializes I/O BARs and ABAR, and assigns an IRQ to the controller. System BIOS optionally sets up an ISR for the SATA controller and enables interrupts for the SATA controller.
- System BIOS loads the RAID Option ROM (OROM)

Int13h Drive Access to an ATA Device:

System BIOS Control Flow:

If the System BIOS uses the interrupt mechanism, then the System BIOS ISR gets control.

- If this is an unexpected interrupt, the System BIOS assumes this is a RAID request.
- The System BIOS clears the Interrupt status in the SATA/RAID controller.
- The System BIOS does NOT mask the IRQ in the PIC.
- The System BIOS sets byte 40h:8Eh to non-zero byte value.
- The System BIOS issues an EOI and IRET.

If the System BIOS does not use interrupt mechanism, then the System BIOS does not receive this INT13h request, therefore it does nothing.

Int13h Drive Access to a SATA ATAPI Device:

System BIOS Control Flow:

- System BIOS can choose any method to access the ATAPI drive (interrupt or polling).
- System BIOS completes the INT13h service request.

Contact your Intel Representative to get the RAID Option ROM.

14.1.4.2 Enable Flexible RAID OROM Features

PCH with RAID capable SKU is able to customize the RAID features through setting the Intel® RST Feature Capabilities (RSTF) register before loading the RAID Option ROM. The RAID OROM will enable the desired features based on the setting in that register, please refer to the PCH EDS for more details.

For example, if the platform desired features are RAID0, RAID1, RAID5 and RAID10. System BIOS should set RSTF (ABAR + C8h [15:0]) to 002Fh before loading RAID OROM.

14.1.4.3 RAID UEFI Driver (RST UEFI Driver)

The Intel Rapid Storage Technology RAID UEFI Driver is an UEFI compliant driver that installs protocols necessary for successful IO operations with integrated AHCI controller in RAID mode (aka RAID controller). The driver is also responsible for publishing HII menu in system BIOS to deliver user interface for the RAID implementation. The UEFI driver is used up to the point where the PCH RAID OS driver takes over.

The RAID UEFI Driver is compiled with UDK 2010 framework and is delivered as efi file. It uses UEFI infrastructure through pointer to the UEFI System Table which provides necessary services for applications/drivers working in UEFI firmware environment (e.g. memory allocation).

Initialization:

System BIOS configures the SATA controller for RAID mode, initializes I/O BARs and ABAR.

The BIOS must implement the following UEFI 2.3.1C protocols/functions:

EFI_BOOT_SERVICES:

LocateHandleBuffer

OpenProtocol

CloseProtocol

WaitForEvent

HandleProtocol

FreePool

AllocatePages

AllocatePool

InstallMultipleProtocolInterfaces

UninstallMultipleProtocolInterfaces

Stall

EFI_RUNTIME_SERVICES:

SetVariable

GetVariable

GetTime

EFI HII PROTOCOLS:

Form Browser 2 Protocol

Config Routing Protocol

HII String Protocol

HII Database Protocol

EFI_ACPI_SUPPORT_PROTOCOL

EFI_ACPI_TABLE_PROTOCOL

All of the listed protocols/functions are required by the Intel RST UEFI Driver.

Access to an ATA device

RAID UEFI Driver installs Block IO protocol which handles all read and write requests to ATA devices.

Access to an ATAPI device

RAID UEFI Driver installs Block IO protocol which handles all read requests to ATAPI devices.

14.1.5 Power Optimizer Configuration

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

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

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

For Skylake/Kaby Lake PCH-LP:

1. Set SATA SIR Index 64h[15:0] = 9001h
2. Set SATA SIR Index 66h[15:0] = 883Ch
3. Set SATA SIR Index 68h[15:0] = 880Ah
4. Set SATA SIR Index 60h[3, 0] = [1b, 1b]
5. Set SATA SIR Index 60h[1] = 1b
6. Set SATA SIR Index 50h[18, 17, 16] = 1s to enable SATA MPHY dynamic power gating.
7. Set SATA SIR Index 54h[30, 26, 25, 24, 10, 9, 8] = 1s
8. Set SATA SIR Index 5Ch[18, 17, 16] = 1s
9. Set SATA SIR Index 70h[26, 25, 24, 15, 10, 9, 8] = 1s, [23:18] = 010110b
10. Set SATA SIR Index 6Ch[29:24] = 101100b, [21:16] = 011110b, [12:8] = 10001b, [4:0] = 01000b

For Skylake / Kaby Lake PCH-H:

1. Set SATA SIR Index CCh[15:0] = 9001h
2. Set SATA SIR Index CEh[15:0] = 883Ch
3. Set SATA SIR Index D0h[15:0] = 880Ah
4. Set SATA SIR Index C8h[3, 0] = [1b, 1b]
5. Set SATA SIR Index C8h[1] = 1b
6. Set SATA SIR Index 90h[7:0] = 1s to enable SATA MPHY dynamic power gating.
7. Set SATA SIR Index 84h[23:16, 7:0] = 1s
8. Set SATA SIR Index A4h[14] = 1b
9. Set SATA SIR Index A8h[18, 17, 16] = 1s
10. Set SATA SIR Index 8Ch[23:16, 7:0] = 1s
11. Set SATA SIR Index A0h[15] = 1b, [23:18] = 010110b
12. Set SATA SIR Index D4h[29:24] = 101100b, [21:16] = 011110b, [12:8] = 10001b, [4:0] = 01000b

If the SATA port has one or more of the following, BIOS must disable MPHY Dynamic Power Gating by set PHYDPGE = 0 and disable Squelch Circuit by set SQOFFIDLED = 1. Please refer to Skylake/Kaby Lake EDS for register detail.

- SATA hotplug-enabled port (PxCMD.HPCP=1)
- SATA external port (PxCMD.ESP=1)
- SATA Slimline port with zero-power ODD (ZPODD) attached (or other AN capable ODD)

14.1.6 Additional Programming Requirements during SATA Initialization

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

For Skylake / Kaby Lake PCH-LP:

1. System BIOS must set SATA PCI offset 94h[8:0] = 183h as part of the chipset initialization prior to SATA configuration.
2. Set OOB Retry Mode bit of Port Control and Status (PCS) register (SATA PCI offset 92h[15]). This should be done regardless of the mode the SATA controllers are operating in
3. System BIOS must program SATA HSIO table BEFORE the SATA ports are enabled.
4. If SATA is disabled or no lanes assigned to SATA when in AHCI mode, then perform the disabling steps to function disable the SATA Controller

5. Program SATA PCI offset 98h[29, 23, 22, 20, 19, 18, 12:7, 6:5, 3:2] to [1, 1, 1, 1, 1, 1, 4, 1, 1]
   a. Note: bit 22 is optional for server platform
6. Program SATA PCI offset 9Ch[5] to 1b (Note: this must be programmed together with SATA PCI offset 9Ch[7:6] in word write)
7. Program SATA PCI offset 9Ch[31] to 1b at the End of Post

For Skylake/Kaby Lake PCH-H:

1. System BIOS must set SATA SIR Index 98h[8:0] = 183h as part of the chipset initialization prior to SATA configuration.
2. System BIOS must set SATA SIR Index A4h[6] = 1b. This should be done regardless of the mode the SATA controllers are operating in
3. System BIOS must program SATA HSIO table BEFORE the SATA ports are enabled.
4. If SATA is disabled or no lanes assigned to SATA when in AHCI mode, then perform the disabling steps to function disable the SATA Controller
5. Program SATA SIR Index 9Ch[29, 23, 20, 19, 18, 12:7, 6:5, 3:2] to [1, 1, 1, 1, 1, 4, 1, 1]
   a. Note: bit 22 is optional for server platform
6. Program SATA PCI offset 9Ch[5] to 1b (Note: this must be programmed together with SATA PCI offset 9Ch[7:6] in word write)
7. Program SATA PCI offset 9Ch[31] to 1b at the End of Post

14.1.7 External SATA Programming

An external SATA port is defined as a user accessible port for the use of an external device that resides outside the system chassis. It has its signal connector externally accessible as a signal only connector (i.e. power is not part of that connector). A specific connector and cable are defined in the SATA Specification for external ports. If external SATA port is implemented in the system, there are additional steps that need to be followed.

- Follow steps in 14.1.6 for additional programming to external SATA port
- Enable the port thru the corresponding SATA PCI offset 92h[n:0] PxE bit.
- Put the port into Listen Mode (refer to AHCI spec for Listen mode info) which achieves similar power savings as if the port is in SATA Slumber power state.
- Set CAP.SXS (Supports External SATA) to ‘1’.
- For each external port set PxCMD.ESP (External SATA Port) to ‘1’ if the port has the specific external port connector. Otherwise insure that this field is ‘0’ (default).

14.1.8 Compliance to Industry Specifications

To ensure a satisfactory user experience and to provide the RAID option ROM implementation with a ‘well-known’ framework under which it can operate, the system BIOS and option ROM must be implemented in accordance with the following specifications:

- The System BIOS must be compliant with the Enhanced Disk Drive Specification, v3.0, rev0.8.
- The System BIOS must be compliant with the BIOS Boot Specification, v1.01.
- The System BIOS must be compliant with the PCI BIOS Specification, v2.1.
- The System BIOS must implement the Post Memory Manager (PMM) services and must be compliant with the POST Memory Manager Specification, v1.01.
- The System BIOS must be compliant with the Plug and Play BIOS Specification, v1.0A.
- The option ROM must implement all required headers (PnP Option ROM and PnP Expansion) and structures (PCI Data Structure) in accordance with the BIOS Boot Specification, v1.01.
- The System BIOS must be compliant with the SATA AHCI Revision 1.31 specification.

14.2 DevSleep Enabling

PCH from Lynx Point LP onwards 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, etc.).

During system POST, some additional activities need to be completed by BIOS during storage initialization. For each implemented port, BIOS must 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 if the Identify Device Data 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 if the Identify Device Data log (30h, 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 must 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.

Note: BIOS must 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.

Note: Step 5 to 11 should be implemented in AHCI driver of System BIOS as it is not covered by PCH reference code. For RAID configuration, it must be supported by the OS driver. On Windows*, this will be supported by the Intel RST driver.

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.

For Skylake/Kaby Lake PCH-LP:

1. Set PxE, PCI offset 92h [3:0] to 0.
2. Clear “Ports Implemented”, ABAR + Ch to 0.
3. Clear SATA MSE and IOE, SATA PCI offset 4h[1:0] to 0.
4. Set SATA Port Clock Disable bits PCI offset 94h[26:24] to all 1’s.
5. Enable all SATA dynamic clock gating and dynamic power gating features
   a. SATA PCI offset 98h bit [29, 23] = [1, 1]
   b. SATA SIR offset 70h bit [26:24, 15, 10:8] = [1s, 1, 1s]
   c. SATA SIR offset 54h bit [30, 26:24, 21] = [1, 1, 1s, 1]
   d. SATA SIR offset 58h bit [26, 25, 24] = [0, 0, 0]
   e. SATA SIR offset 5Ch bit [17, 16] = [1, 1]
   f. For each RST PCIe Storage Cycle Router, program the "Extended General Configuration Register" field, PCI offset 354h[20, 19, 17, 16] = 1b, 1b, 1b, 1b
6. Disable SATA Device by setting SCFD, PCI offset 9Ch[10] to 1.
7. Set PCR[PSF_1] + "AGNT_T0_SHDW_PCIEN"[SATA][8] to 1.

Note: For Skylake PCH-LP, while SATA is located at PSF_4, its PSF shadowed registers are actually implemented in PSF_1

Note: SIR-based registers are accessed through SATA PCI offset A0h/A4h register pairs.

For Skylake/Kaby Lake PCH-H:

1. Set PxE, PCI offset 94h [7:0] to 0.
2. Clear “Ports Implemented”, ABAR + Ch to 0.
3. Clear SATA MSE and IOE, SATA PCI offset 4h[1:0] to 0.
4. Set SATA Port Clock Disable bits PCI offset 90h[7:0] to all 1’s.
5. Enable all SATA dynamic clock gating and dynamic power gating features
   a. SATA SIR offset 9Ch bit [29, 23] = [1, 1]
   b. SATA PTM2, “SIR offset 84h” bit [23:16] to [FFh]
   c. SATA PTM4, “SIR offset 8Ch” bit [23:16, 7:0] to [FFh, FFh]
   d. SATA CTM3, “SIR offset A0h” bit [15] to 1
   e. SATA CTM4, “SIR offset A4h” bit [26, 25, 24, 14, 5] to [0, 0, 0, 1, 1]
   f. SATA CTM5, “SIR offset A8h” bit [17, 16] to [1, 1]
   g. For each RST PCIe Storage Cycle Router, program the "Extended General Configuration Register" field, PCI offset 354h[20, 19, 17, 16] = 1b, 1b, 1b, 1b
6. Disable SATA Device by setting SCFD, PCI offset 9Ch[10] to 1.
7. Set PCR[PSF_1] + "AGNT_T0_SHDW_PCIEN"[SATA][8] to 1.

§ §

15 High Performance Event Timer (HPET) Support

PCH includes a High Performance Event Timer (HPET) implementation of 1 timer block with 8 comparators. The System BIOS reports to the OS the location of the register space by using the ACPI 3.0 mechanism. There are 4 possible memory address ranges beginning at:

- FED0_0000h
- FED0_1000h
- FED0_2000h
- FED0_3000h

The choice of these address ranges as well as the enable of the address range is selected by the HPTC Register, P2SB PCI offset 60h[7, 1:0].

From Skylake PCH onwards, clock frequency is changed from 14.318MHz to 24 MHz. For code backward compatibility, can refer to COUNTER_CLK_PER_CAP register field for the main counter tick rate.

Refer to the PCH EDS for more details and for a description of the operating modes.

15.1 High Performance Event Timer Description Table (HPET)

The System BIOS 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 to the IA-PC HPET (High Precision Event Timers) Specification.

Note: Because of the clock frequency change, there is a need to update the “Main Counter Minimum Clock_tick in Periodic Mode” entry within the HPET table.

15.2 HPET ASL CODE

The following ASL code is required by the System BIOS to report resource consumption for HPET:

```
Device (HPET)
{
         Name(_HID, EISAID"PNP0103")
                   // Newly assigned PNPID for IAPC HPET

         Name(_UID,<n>)
                            // used if there are more than 1 timer blocks,
                            // where <n> is an integer unique to each Device
                            // returning the same _HID value

         Name(Buffer, BUF0)
{                  // buffer object containing timer resource descriptors

         {<memory resource descriptor>}
                   // Report 1kB of memory consumed by this timer block

         {<interrupt resource descriptor>}
                   // Optional: only used if BIOS allocates interrupts
}
         Method(_CRS,0)
{        // method to return resource descriptors
         Return(BUF0)
}
```

§ §

16 GPIO Handling

Most of the GPIOs are multiplexed with other alternate functions and default to their alternate functions on reset. The System BIOS must select each to be either native (alternate function) or GPIO after reset. If it is selected as GPO, BIOS will need to program the polarity for the particular platform application. The PCH EDS contains details of the register information for GPIO programming and native function descriptions.

From SKL PCH onwards, the GPIO registers are under the private configuration register space, accessible through the P2SB, using either SBI or MMIO method.

Some GPIO bits might reside in primary well, while others might be in DSW well and their reset behavior is further controlled by the "Pad Reset Config". Please see individual bit definition in EDS for more detail.

16.1 ACPI Compliant GPIO

All GPIs are "ACPI Compliant," meaning their Status and Enable bits reside in ACPI I/O space. In Skylake/Kaby Lake PCH, there are GPIs that have 1:1 mapping (1-tier) to the GPE0 enable/status, and there are GPIs that are grouped under one master GPE0 enable/status and need 2 level enable/status (2-tier) within GPIO register space.

It is important to understand that the various GPIs have different levels of functionality when used as wake events. For example, the GPIs that reside in the primary power well can only support ACPI runtime event and Sx wake, but not the Deep Sx wake that requires DSW well GPIs.

Any of the GPIOs can be enabled to trigger wake, but only those status of GPIO mapped to 1-teir scheme are directly accessible through the GPE status registers. For those GPIO mapped under 2-tier scheme, their status would be reflected under single master status, "GPIO_TIER2_SCI_STS" bit ABASE offset 8Ch[15] or GPE0_STS [6Fh] and further comparison needed to know which 2-tier GPI(s) has triggered the GPIO Tier 2 SCI.

16.1.1 1-Tier GPEs

GpioPads which are within groups that are in 1-Tier GPE have exclusive GPE event number. Only three GpioGroups can be configured for 1-Tier GPE. Those groups GPE status and enable register will be mapped to ACPI IO Space (GPE0_STS[95:0], GPE0_EN[95:0]). To set GPIO groups for 1-Tier GPEs following registers need to be programmed:

- PMC.GPIO_CFG.GPE0_DW2 and GPIO.MISCCFG.GPE0_DW2 (for all communities)
- PMC.GPIO_CFG.GPE0_DW1 and GPIO.MISCCFG.GPE0_DW1 (for all communities)
- PMC.GPIO_CFG.GPE0_DW0 and GPIO.MISCCFG.GPE0_DW0 (for all communities)

It is important to know the ACPI _LXX method is depending on this mapping. For platforms which have Embedded Controller, SCI signal from it must be connected to GpioPad which is in 1-Tier so that it has exclusive GPE number. This requirement is mentioned in ACPI cspec.

16.1.2 2-Tier GPEs

All GPIO groups that are not in 1-Tier are in 2-Tier GPE. For those groups all GPE status bits are ORed and the result is in PMC.GPE0_STS_127_96[15].GPIO_TIER2_SCI_STS. All pads from 2-Tier share one GPE event - GPE111.This event will appear under _L6F method in ACPI.

16.1.3 Example GPE configuration

BIOS has to make sure besides the PMC register programming, it also needs to update MISCCFG PCR[GPIO] + offset 10h under all GPIO communities with the exact same mapping.

Example 1 shows how it can be determined GPI to GPE Mapping for certain 1-Tier GPIO pad SCI:

For the GPIO pad SCI below:

PMC.GPIO_CFG.GPE0_DW2 and GPIO.MISCCFG.GPE0_DW2 = GPP_C

PMC.GPIO_CFG.GPE0_DW1 and GPIO.MISCCFG.GPE0_DW1 = GPP_B

PMC.GPIO_CFG.GPE0_DW0 and GPIO.MISCCFG.GPE0_DW0 = GPP_A

Then 1-Tier events would be assigned as follow:

GPE[95:88] = not used

GPE[87:64] = GPP_C[23:0]

GPE[63:56] = not used

GPE[55:32] = GPP_B[23:0]

GPE[31:24] = not used

GPE[23:0] = GPP_A[23:0]

Note: All other groups would fall under 2-Tier GPE 6Fh = 111b

Example 1 GPI to GPE Mapping for certain 1-Tier GPIO pad SCI

Note: While there are 96 bits in the GPE registers to be assigned to GPIOs, there are only up to 72 GPIOs that could have the dedicated GPE mapped because the mapping is done by GPIO group to GPE DWord basis. Currently, PCH has maximum 24 GPIOs per group.

16.1.4 ACPI GPE handling

1-Tier events have exclusive GPE numbers. Methods which will be called by OS during an SCI interrupt will be _Lxx (for level type interrupt) or _Exx (for edge type interrupt.

For example if GPP_A is mapped to GPE0_DW0 then level interrupt from GPP_A_10 will cause _L0A method.

For 1-Tier GPE events there isn't any additional requirement besides appropriate GPIO configuration and providing _Lxx/_Exx method. OS will take care of clearing status bit and enabling SCI in GPE0_EN – it will do so if it sees appropriate handler (_Lxx/_Exx) for this event in ACPI.

For 2-Tier BIOS needs to provide _L6F method and inside this method check what particular GPIO pad caused this event. After handling it needs to clear status bits in GPIO.GPI_GPE_STS. For the Edge trigger events under 2-Tier, the status should be handled carefully and separately to prevent from getting cleared by clearing all status for Lever trigger events.

Example 2-Tier GPE handler:

```
Method(_L6F, 0) // Method which is called for all 2-tier GPE, must be within
_GPE scope
{
  \_SB.UGPS() // Update information on GPIO mapping for 1 and 2 tier GPE
  If (\_SB.ISME(GpioPad))
    {
      \_SB.DXYZ.GPEH() // Custom function to handle GPE for certain GPIO pad,
if it is edge sensitive, it needs to take care of clearing its GPE status bit
    }
  \_SB.CGLS() // Clear all GPE status for 2-tier which are level sensitive
}
```

16.2 GPIO reset configuration

GPIO device has the ability to change the reset type upon which pad configuration will get back to default state. Depending on group type that GPIO pad resides in there are either 3 or 4 different reset settings:

Table 16-1. GPIO reset types

PADCFG_DW0 Pad Reset Config	Platform reset	Is pad reset?	
		GPP	GPD
00 – Powergood	Warm Reset	N	N

PADCFG_DW0 Pad Reset Config	Platform reset	Is pad reset?	
		GPP	GPD
(GPP: RSMRST, GPD: DSW_PWROK)	Cold Reset	N	N
	S3/S4/S5	N	N
	Global Reset	N	N
	DeepSx	Y	N
	G3	Y	Y
01 - Deep GPIO Reset (HOST Deep Reset)	Warm Reset	Y	Y
	Cold Reset	Y	Y
	S3/S4/S5	N	N
	Global Reset	Y	Y
	DeepSx	Y	Y
	G3	Y	Y
10 - GPIO Reset (Platform Reset - PLTRST)	Warm Reset	Y	Y
	Cold Reset	Y	Y
	S3/S4/S5	Y	Y
	Global Reset	Y	Y
	DeepSx	Y	Y
	G3	Y	Y
11 - Resume Reset (GPP:Reserved, GPD: RSMRST)	Warm Reset	-	N
	Cold Reset	-	N
	S3/S4/S5	-	N
	Global Reset	-	N
	DeepSx	-	Y
	G3	-	Y

Not all GPIO Pad configuration settings reset is controllable by this field. There are some registers or fields like MISCCFG, Pad Configuration Lock or Pad Reset Config which reside in Powergood and their reset type cannot be modified.

How certain GPIO Pad reset should be configured highly depends on use case and external pad connection. For example if a GPIO pad is connected to an external signal which is supposed to wake the platform from S3/4/5 state then GPIO reset type must be different than “GPIO Reset”, as with this setting pad configuration (including SCI interrupt settings) would get back to default state during S3/4/5 entering.

16.3 Serial GPIO Feature

PCH is capable of transmitting serial messages across a GPIO. For Skylake/Kaby Lake PCH-only pads 0 to 4 from GPP_D have this ability. The System BIOS can control the rate at which the data is transmitted and the amount of data transmitted (byte, word, or dword) using the Serial Blink, Serial Blink Command Status, and Serial Blink Data registers. See the PCH EDS for a description of the registers. The System BIOS must perform the following steps to utilize this feature.

1. Enable the Serial GPIO feature for the GPIO to be used for the serial data transmission in the GP_SER_BLINK Register bit [4:0].
2. Set the Data Rate in the GP_SER_CMDSTS Register bit [21:16].
3. Set the Data Length by programming GP_SER_CMDSTS Register bit [23:22].
4. Wait until the “Busy” bit is cleared by the hardware, GP_SER_CMDSTS Register bit [8].
5. Write the data to be serialized to the GP_SER_DATA Register.
6. Set the “Go” bit, GP_SER_CMDSTS Register bit [0].
7. Repeat Steps 4 through 6 as necessary.

16.4 GPIO Programming Steps

GPIO pad is controlled by 2 DW registers[5]. Refer to PCH EDS for details.

16.4.1 GPIO Pad in Native Mode

GPIO pads instead of standard GPIO mode provide additional functionality related to different PCH features. To turn it on Pad Mode needs to be switched from GPIO Mode to one of Native modes. What native functionality is available under what pad is described in EDS.

In native modes some settings are not applicable and BIOS doesn’t need to configure them:

- Interrupt related setting:
  - RX Level/Edge configuration,
  - GPIO Input route IOxAPIC/SCI/SMI/NMI,
  - GPI interrupt Enable,
  - GPI General Purpose Event Enable,

[5] When GPIO pads are owned by Intel ME Firmware, BIOS/host should not attempt to access these GPIO Pads registers, registers value returned in this case will be 0xFF.

- o SMI/SCI/NMI enable

- GPIO: Rx Disable, TX state, and Disable.
- Host Software Pad Ownership
- Pad Configuration Lock GPIO TxState

On the other hand the following attributes are still meaningful in native mode and BIOS can configure them if needed.

- Pad Reset Config,
- RX Pad State Select,
- RX Raw override to '1',
- RX Invert,
- Termination

16.4.2 GPIO Pad in ACPI

GPIO pad in ACPI can be used indirectly or explicitly in following way:

- As a GPE event source (please look at paragraph 16.1.4 ACPI GPE handling)
- As an interrupt (please look at paragraph 16.4.3 interrupts)
- Indirectly in other ACPI methods.
- Standard GPIO pin under GPIO OS driver control.

Last case requires special description in ACPI code.

ACPI Example:

```
GpioIo(Exclusive, PullDefault, 0, 0, IoRestrictionOutputOnly,
"\\_SB.PCI0.GPI0",,,GSLP) {GpioIoNumber}
```

GpioIoNumber = GpioPadNumber + GpioGroupNumber * 24

GpioGroup		GpioGroup Number
PCH-LP	PCH-H	
GPP_A	GPP_A	0
GPP_B	GPP_B	1
GPP_C	GPP_C	2
GPP_D	GPP_D	3
GPP_E	GPP_E	4
GPP_F	GPP_F	5
GPP_G	GPP_G	6
---	GPP_H	7

GpioGroup		GpioGroup Number
PCH-LP	PCH-H	
---	GPP_I	8

GpioIo is supplied as one of resources for a device which needs additional GPIO pad control for its own purpose. This GPIO pad will be accessed through GPIO OS driver so HOSTSW_OWN needs to be set for GPIO Driver Mode.

16.4.3 GPIO interrupts

GPIO Pad if has enabled input can trigger an interrupt caused by different input state change. This is configured in GPIO.PADCFG_DW0.RxEvCfg:

- Level (RxInv=0 for ActiveHigh; 1 for ActiveLow)
- Edge (RxInv=0 for rising edge; 1 for falling edge)
- Disabled
- Either rising or falling edge

Depending on Host Software Pad Ownership (HOSTSW_OWN) and GPIO Input Routing (in PADCFG_DW0) settings following interrupts can be generated:

- SCI
- SMI
- NMI
- IOxAPIC interrupt
- GPIO Driver interrupt

For all interrupts pad needs to be in GpioMode (PADCFG_DW0.PMode = Gpio)

16.4.3.1 SCI interrupt

SCI interrupt will generate General Purpose Event under 1 or 2-Tier (please look at paragraph 16.1 ACPI Compliant GPIO). GPIO Group to GPE mapping needs to be first programmed for the system so that it is known within which scheme particular events work. To enable this interrupt:

- GpioPad needs to be routed for SCI interrupt – PADCFG_DW0.GPIRoutSCI = 1
- Pads Host Software Ownership (HOSTSW_OWN) needs to be configured for ACPI Mode, so that GPI_GPE_STS bits are not masked and updated when interrupt triggers

- GPE needs to be enabled for this pad (GPI_GPE_EN[x] = 1). This is required to be done by BIOS for 2-Tier GPEs. For 1-Tier OS will enable them when it sees appropriate ACPI handler for this event (_Lxx/_Exx).

16.4.3.2 SMI interrupt

Only some GPIO pads can be configured to generate an SMI (please look at GPIO documentation). To make certain pad an SMI source following configuration needs to be applied:

- GpioPad needs to be routed for SMI interrupt – PADCFG_DW0.GPIRoutSMI = 1
- Pads Host Software Ownership (HOSTSW_OWN) needs to be configured for ACPI Mode, so that GPI_SMI_STS bits are not masked and updated when interrupt triggers
- SMI needs to be enabled for this pad (GPI_SMI_EN[x] = 1).

Besides that last qualifier (GPI_SMI_EN) will be set during registration of particular GPI SMI handler through `EFI_SMM_GPI_DISPATCH2_PROTOCOL`.

16.4.3.3 NMI interrupt

Only some GPIO pads can be configured to generate an NMI (please look at EDS). To make certain pad an NMI source following configuration needs to be applied:

- GpioPad needs to be routed for NMI interrupt – PADCFG_DW0.GPIRoutNMI = 1
- Pads Host Software Ownership (HOSTSW_OWN) needs to be configured for ACPI Mode, so that GPI_NMI_STS bits are not masked and updated when interrupt triggers
- NMI needs to be enabled for this pad (GPI_NMI_EN[x] = 1).

16.4.3.4 IOxAPIC interrupt

Each GPIO Pad can be configured to directly generate an IRQ in APIC using following configuration:

- GpioPad needs to be routed for IOxAPIC interrupt – PADCFG_DW0.GPIRoutIOxAPIC = 1
- Pads Host Software Ownership (HOSTSW_OWN) is recommended to be configured for ACPI Mode although this is not a strict requirement. ACPI mode would be set here for clarity that this pad is not under ownership by GPIO OS Driver
- GpioPad reset needs to be set to 10b – “GPIO Reset” (PLTRST) so that GPIO configuration is reset with the same signal as ITSS interrupt controller

Care needs to be taken because IRQn number that pad will generate is not configurable. Interrupt number can be read from PADCFG_DW1.IntSel register or from EDS and is within 24-119 range. Any interrupt above IRQ23 is not shareable so if certain pad is enabled for IOxAPIC interrupt it needs to be checked if no other GPIO or device uses this interrupt number.

For the OS to handle this interrupt additional ACPI entry needs to be supplied so that OS knows which driver it needs to notify about the interrupt:

ACPI Example:

```
Interrupt(ResourceConsumer, Level, ActiveLow, ExclusiveAndWake,,,)
{IrqNumber}
```

16.4.3.5 GPIO Driver interrupt

GPIO device can also generate one APIC interrupt for all available GpioPads. When this interrupt is triggered, OS notifies GPIO OS driver which then needs to further distinguish which GpioPad is the source of it. Configuration for this interrupt is done in following way:

- Choose IRQ number in MISCFG.GPIO_Driver_IRQ_Route between IRQ14 (default) and IRQ15. The same setting needs to be applied for all communities
- Pads Host Software Ownership (HOSTSW_OWN) needs to be configured to GPIO Driver Mode, so that GPI_INT_STS bits are not masked and updated when interrupt triggers
- GPI_INT_EN needs to be enabled for this pad (GPI_INT_EN[x] = 1).

GPI_INT_EN will be enabled by GPIO OS Driver, when GpioInt ACPI description is supplied:

ACPI Example:

```
GpioInt (Level, ActiveLow, ExclusiveAndWake, PullDefault, 0x0000,
"\\_SB.PCI0.GPI0", 0x00, ResourceConsumer,) {GpioIntNumber}
```

GpioIntNumber - Refer to Table TBD for the calculation method.

The value of main GPIO device interrupt needs to be supplied in GPIO Device ACPI resources description (refer to TBD), so that OS knows that it needs to notify GPIO OS driver when it detects this interrupt (IRQ14/15).

When GPIO pad is correctly configured for GPIO Driver interrupt at OS level the flow after interrupt is triggered is following:

GPIO pin state changes -> GPIO Device IRQ14/15 is triggered -> OS detects IRQ14/15 -> OS notifies GPIO OS Driver -> GPIO OS Driver looks at GPI_INT_STS/EN register to determine the GpioPad source -> Device which has GpioInt ACPI description in it is notified about the interrupt.

16.4.4 Termination

Termination setting can be configured in PADCFG_DW1.Term register. What type of termination is applied highly depends on GPIO pad usage and board schematics. Native termination setting is only available for some GPIO pins (please check EDS). With this setting native functionality has control over termination.

16.4.5 Not connected pad recommended settings

If certain pad is not used neither in GPIO mode nor in any native functionality then recommended GPIO configuration for this pad is:

- Set Pad Mode to GPIO mode
- Disable both input and output
- Leave other settings with default hardware value

16.4.6 Pad Mode Enabling for PWM Mode

This enabling is applicable for GPP_Dx.

Below steps are needed to initialize PWM mode:

- Configure GPP_Dx to PadMode = 2 (SKL/KBL PCH-LP) or 4 (SKL/KBL PCH-H) as Blink mode (PWM mode).Configure the PWM frequency in PWM control, look into EDS 27.2.46 of GPIO COMM1 register.
    - PWM programming sequence
        - Initial Enable or First Activation
            - Program the Base Unit and On Time Divisor values
            - Set the Software Update Bit
            - Enable the PWM Output by setting the PWM Enable bit
            - Repeat the above steps for the next PWM module
        - Dynamic update while PWM is Enabled
            - Program the Base Unit and On Time Divisor values
            - Set the Software Update Bit
            - Repeat the above steps for the next PWM module

The PWM base freq is 32.768KHz, so the PWM output frequence is :

OutputFrequency = (32.768KHz) * (base_unit_value[6] / 256)

16.5 GPIO Pad Ownership

GPIO pad can be owned by one of following components:

[6] The base unit value is unsigned 8 integer bit and 14 fraction bit in base unit.

- Host
- CSME
- ISH

Ownership configuration is in PAD_OWN register. This setting is RO for host and configurable by CSME. If pad ownership is set for CSME/ISH then host (e.g. BIOS) cannot access this pads settings. Reading from host GPIO register (except PAD_OWN) for a pad owned by CSME/ISH will return FFFFFFFFh. CSME can change ownership of a pad based on enabled features for CSME or ISH.

16.6 GPIO Registers Lockdown

PCH includes GPIO register lock-down feature, which allows the GPIO registers to be locked down and Read-Only when the relevant Pad Configuration Lock bit is set. To unlock the GPIO registers, the relevant Pad Configuration Lock bit must be cleared. "Pad Configuration Lock" can only be set or cleared using opcode 13h, with non-posted sideband write.

There is a separate lock control for the GPIOTxState through the relevant "Pad Configuration Lock GPIO TxState" bit which has exact requirements as above.

If GPIO configurations are not used after boot, it is recommended that the relevant "Pad Configuration Lock", "Pad Configuration Lock GPIO TxState" and the PMC.GPIO_UNLOCK_SMI_EN bits are set by BIOS prior to end of POST. Refer to the PCH EDS for more details and a list of registers that are locked down. Once lock is performed and PMC.GPIO_UNLOCK_SMI_EN set, SMI will be generated if lock is tried to be cleared. This ensures that only SMM code can change the above GPIO registers after the lockdown.

Because the P2SB SBILOCK would be set during post thus there is no mechanism to unlock the GPIO during runtime.

Lock down registers reset type is hardcoded to Powergood and not controllable by PADCFG_DW0.PadRstCfg. Care needs to be taken when locking a pad with different reset type. If not handled properly situation may occur in which after G3 (when all registers are reset to default) desired GPIO configuration is provided but after some power state transition (like Sx) most of the pad configuration got to default state. Unfortunately lock will still work and prevent programming of pad configuration (please look at paragraph TBD for reset details). There are two solutions to handle this:

- For the pad that is locked set reset type to Powergood so that GPIO configuration and lock bit setting get back to default state at the same time
- Unlock GpioPad before it is configured and lock it. If this is done at early BIOS stage before SMIs are enabled, non-SMM code can do the unlock.

16.7 GPIO VISA/PTI Pin Support

GPIO can be configured to support VISA/PTI function. It is expected that the VISA setup tool would lock the GPIO pad after its setup so that BIOS could not mess with

the setup. Thus BIOS should not panic when finding pad being locked, and pad register read is not per what being written (when locked).

16.8 GPIO Pad Special Handling

16.8.1 PCH-LP GPP_A_12

For security reasons, if the pad is host-own, BIOS has to always lock this pad after being initialized. GPP_A_12 has a Native3(SX_EXIT_HOLDOFF#) mode which allows to delay resuming to S0. If this pad is not locked and platform was not initialy designed for this functionality, malware could reconfigure this pads setting under OS (switch to Native3) which would make platform not able to resume until G3 is applied.

It is ok not to lock its Tx. By this requirement, it means that this pad will not be allocated to GPIO driver usage and must not have any runtime configuration change (except Tx) requirement.

16.8.2 General Purpose SPIs' Chip Select Pads

For both Skylake PCH-LP and Skylake PCH-H, these pins are GPP_B_15 and GPP_B_19. If any of the pad is to be used as chip select function, the related pad mode programming must not be done until a needed programming is done. Refer to TBD section (22.5.)

Note: Intel BIOS Reference Code will delay the related pads' pad mode programming till Serial I/O SPI initialization.

16.8.3 SATA DevSlp

For both Skylake /Kaby Lake PCH-LP and Skylake/Kaby Lake PCH-H, when its pads are configured to support DevSlp native, its PadRstCfg field is also to be programmed to 00b, Powergood. Refer to section 14.2.1.

Note: The DevSlp pin also has Sx isolation need as stated in below section, where such programming is only carried out within Sx handler, compare to this requirement that is done during post time. Because of these two different requirement, the GPIO that is configured to DevSlp native, must not have its pad configuration locked

16.9 GPIO Pin Isolation During Sx

Devices on the board may reside in traditional CORE/ASW power wells and PCH logics are powered by the Primary Power Well which lives on in both S0 and Sx power states. PCH inputs driven from such devices and PCH outputs to such devices need proper isolation in Sx states.

It is expected the platform would describe each pad's connection, whether it's device pad under Always On rail (suswell, DSW) or S0 rail (Core well). Based on the connection, the BIOS would perform the needed Sx Isolation accordingly.

Note: If there is no connection at all, declare that pin connection as non-Core well.

Pad-mode Default/configured	BIOS GPIO Sx Isolation Action
Default GP-In *exclude if configured to SRCCLKREQB_#	Set PadRstCfg to 10b (PLTRST#)
Default Gp-Out	Set PadRstCfg to 10b (PLTRST#)
Configured SRCCLKREQB_0/1/2/3/4/5	Set PadRstCfg to 00b (RSMRST#)
Default EXT_PWR_GATEB	Set PadRstCfg to 10b (PLTRST#)

Note: If the pad is locked at post time, the related pads' PadRstCfg field could be configured accordingly during post, else the additional similar handling would be done in S3 SMI handler.

Refer to PCH EDS for details handling requirement on each pin that connect to core well devices.

16.10 GPIO Device ACPI Requirement

Declare the GPIO device as ACPI device for GPIO driver usage.

```
External (\SBG0) //SBREG_BAR + (gpiocom0 << 16)
External (\SBG1) //SBREG_BAR + (gpiocom1 << 16)
External (\SBG3) //SBREG_BAR + (gpiocom3 << 16)

      Device (GPI0)
      {
         Method (_HID) {
           // Return HID based on PCH Product
           if(LEqual(PCHV, SPTH)){ Return("INT345D") }
           Return("INT344B")
        }
        Name (RBUF, ResourceTemplate ()
        {
          Memory32Fixed (ReadWrite, \SBG0, 0x00010000, BAR0)
          Memory32Fixed (ReadWrite, \SBG1, 0x00010000, BAR1)
          Memory32Fixed (ReadWrite, \SBG3, 0x00010000, BAR3)
          Interrupt (ResourceConsumer, Level, ActiveHigh, Shared, , ,
INT0) { 14 }
        })

        Method (_CRS, 0x0, NotSerialized)
        {
          Return (RBUF)
        }

        Method (_STA, 0x0, NotSerialized) {
          If(LEqual(GPEN, 0))   { Return(0x0) }
          Return(0xF)
        }
      }   //  Device (GPI0)
```

§ §

17 Thermal Management Support

PCH has a built-in internal thermal sensor. With this thermal sensor, the thermal subsystem device, external thermal sensors from other onboard components, Intel® ME firmware and BIOS support, PCH can support combination of features listed below:

- Thermal Reporting (TR)
- Intel® Dynamic Platform and Thermal Framework (DPTF)

There are many different hardware and software components required for the platform to support the features above. This chapter outlines some general BIOS requirements for the above features; please refer to the specific BIOS specifications for more details.

17.1 Thermal Management Features Overview

Thermal Reporting (TR) allows SMBus compatible devices such as Embedded Controller (EC) or SIO to obtain system thermal data from PCH via the SMLink1 interface. It is implemented by using various thermal sensors and integrated thermal subsystem device with support from Intel® ME firmware and BIOS. The thermal data provided can be used by EC to maintain thermal control policy or allow host software to display system thermal info.

Intel® Dynamic Platform and Thermal Framework (Intel® DPTF) is envisioned as a platform level software framework that serves as a base for any power, thermal or other platform level control policy to be implemented and deployed. The Dynamic Power Performance Management (DPPM) is a platform level closed loop thermal management solution that utilizes the Intel® DPTF infrastructure to monitor and control device temperatures on the platform.

17.2 Thermal Subsystem Device Initialization

The System BIOS must perform the following steps to initialize the PCH thermal subsystem device.

1. Refer to section 6.1.2 for interrupt assignment requirement
2. Go through general PCI enumeration and assign standard PCI resource. It is recommended not to setup TBARB and TBARBH, as there is no current runtime use case of TBARB. System BIOS must use TBAR instead for the thermal subsystem device initialization.
   a. Use TBAR instead of TBARB, could be used by BIOS SMI if needed, especially once the thermal device is hidden.
   b. If TBARB is to be used, we need program the DMI register for source decode. Refer to section 2.5.1.1
3. Initialize relevant Thermal subsystems for the desired features.
4. If we need to expose the thermal subsystem device as a PCI device for mobile platforms that support Intel DPPM, make sure the device is not hidden.

5. Else if there is no runtime usage, BIOS needs to perform the following steps prior to end of POST to free up the PCI resources and hide the thermal subsystem device
   a. Clear the Memory and Bus Master enable bit
   b. Release and clear memory resource assigned in TBAR (PCI offset 10h-13h)
   c. Hide PCH thermal device PCI configuration space by setting
      i. Thermal Controller Function Disable (TCFD) [0]
      ii. PCR[PSF_2] + "PCI Configuration Disable"[Thermal][0].
6. Else if BIOS needs runtime access to PCH thermal subsystem device data.
   a. Hide PCH thermal device PCI configuration space by setting
      i. Thermal Controller Function Disable (TCFD) [0]
      ii. PCR[PSF_2] + "PCI Configuration Disable"[Thermal][0]
   b. Ensure TBAR memory is reserved and reported to the OS as motherboard resources to avoid memory allocation conflicts.

17.3 Thermal Sensor Initialization

This section describes initialization for various thermal sensors in Intel products. If DIMM does not have thermal sensor, then the required/recommended below become irrelevant.

Table 17-1: Thermal Management Sensor Enabling

	PCH	DIMM
TR	Recommended	Recommended
DPPM	Required	Required

17.3.1 Initializing Thermal Sensors

The System BIOS must perform the following steps to initialize the thermal subsystem device. The System BIOS is required to repeat this process on a resume from Sx. BIOS may enable any or all of the registers below based on OEM's platform configuration. Intel does not recommend a value on some of the registers, since each platform has different temperature trip points and one may enable a trip to cause an SMI while another platform would cause an interrupt instead.

The recommended flow for enabling thermal sensor is by setting up various temperature trip points first, followed by enabling the desired trip alert method and then enable the actual sensors from TSEL registers. If this flow is not followed, software will need to take special care to handle the false events during setting up those registers.

1. Set various trip points based on the particular usage model. Note that Cat Trip must always be programmed.

- Trip Point Temperature = (Trip Point Register [8:0]) / 2 – 50 °C

- The thermal sensor provides an accuracy of +/-5°C over the temperature range from 50°C to 110°C, +/- 7°C over 30°C to 50 °C and +/- 10 °C over -10 °C to 30 °C. The sensor is functional, from -40 °C to -10 °C and from 110 °C to 130 °C, but the accuracy is not specified for that range.

- CTT must be programmed for Cat Trip, recommended to program TBAR + 10h[15:0] to 14Ah, CTT must never be changed while the TS enable is set. This rule prevents a spurious trip from occurring and causing a system shutdown. TSC must then be written to 81h to enable the power down and lock the register.

- TAHV and TAHL may be programmed if the BIOS or driver wish to force a SW notification for PCH temperature

  - If TAHL/TAHV programmed much later in the flow when a driver is loaded, this means that the TS had been enabled long before this, the thermal sensor must be disabled when TAHL/TAHV are programmed, and then re-enabled.

  - TSPIEN or TSGPEN may be programmed to cause either an interrupt or SMI/SCI.

  - It is recommended that TAHV, TALV, TSPIEN and TSGPEN be left at their default value, unless there is a specific usage that requires these to be programmed.

2. Clear trip status from TSS/TAS. BIOS should write FFh to clear any bit that was inadvertently set while programming the TS. This write of FFh should be done before continuing to the next steps.

3. Enable the desired thermal trip alert methods, i.e. GPE (TSGPEN), SMI (TSMIC) or Interrupt (TSPIEN). Only one of the methods should be enabled and the method will be depending on the platform implementation.

   - TSGPEN: BIOS should leave this as default 00h, unless it is required to enable GPE.

   - TSMIC: BIOS should leave TSMIC[7:0] as default 00h, unless the SMI handler is loaded and it's safe to enable SMI for these events.

   - TSPIEN: BIOS should leave this as default 00h, so that a driver can enable later.

4. Enable thermal reporting to an EC over SMBus or eSPI and PMC.

   a. Write 01h to TSREL.

5. If the PCH_Hot pin reporting is supported, then write the temperature value and set the enable in PHL.

Note: For PCHHOT# support, we need to make sure if GPP_B_23 is set to PCHHOTB mode (pad mode == 2).

6. If thermal throttling is supported, then set the desired values in TL. It is also being used for PCH-processor cross throttling support, see section 17.5.

7. Enable thermal sensor power management.

   a. Write C8h to TSPM[8:0]

b. Write 101b to TSPM[11:9]

c. Write 100b to TSPM[14:12]

d. Lock the register by writing ‘1’ to TSPM[15]

8. For LP:
   Program TBAR + C0h [0] = 1b
   Program TBAR + C4h[12:8, 7:6] = 00111b, 11b
   Program TBAR + D0h[25:21] = 00101b
   Program TBAR + E4h[28:27, 26:24, 11:8, 4:1] = 11b, 001b, 1000b, 1000b
   Program TBAR + E8h[15:8] = 80h
   If TBAR + CCh[15:0] is 0h, then program TBAR + E0h to 800002BEh
   For H:
   Program TBAR + C0h [0] = 1b
   Program TBAR + C4h[12:8, 7:6] = 00111b, 11b
   Program TBAR + C8h[31:16] = 0010h
   Program TBAR + D0h[25:21] = 01000b
   Program TBAR + E4h[28:27, 26:24, 11:8, 4:1] = 11b, 001b, 1000b, 1000b
   Program TBAR + E8h[15:8] = 80h
   If TBAR + CCh[15:0] is 0h, then program TBAR + E0h to 800002BEh
9. Program TBAR + A4h[2, 1, 0]= 1b, 1b, 1b
10. Then Enable thermal sensor by programming TSEL register to 01h. This should be done after the steps above.
11. Lock down the thermal reporting to prevent outside agents from changing the values.

    a. It is recommended that TSC [7] set to ‘1’ to lock the CAT Trip behavior. This lock also locks CTT.

    b. TSMIC [7] locks SMI reporting of thermal events

    c. PHLC [0] locks the PHL and PHLC registers for PCH_Hot#

    d. TL [31] locks the thermal throttling registers

    e. TL2[15] locks the thermal throttling 2 register (see below step)

    f. TSEL [7] locks the thermal sensor enable, after TAHV and TAHL are programmed by BIOS or driver later in case.

17.4 Thermal Reporting Requirements

To support a platform EC that is managing the system thermals, the PCH provides the ability for the EC to read the PCH temperature over SMBus and over eSPI. This entails having the thermal sensor unit drive the temperature directly to the SMBus and eSPI units. The EC will issue an SMBus read or eSPI OOB channel request and receives a single byte of data, indicating a temperature between 0 °C and 254 °C, where 255 (FFh) indicates that the sensor isn’t enabled yet. The EC must be connected to SMT3 (SMLink1) or eSPI for thermal reporting support.

Upon reset, the value driven to the EC will be FFh. This indicates that BIOS has not enabled the reporting yet. When the EC receives FFh for the temperature, then it knows that the thermal sensor is not enabled and can assume that we are in the boot phase and that power is low or that by default during boot the cooling should be increased since the actual temperature is unknown. After being enabled, the EC will receive a value between 0h and 7Fh (0 °C to 127 °C). If the EC ever sees a value

between 80h and FEh, then an error has occurred, since the PCH should have shut down the platform before the temperature ever reached 128 °C, since cat trip for the PCH will always be below 128 °C. The PCH itself doesn't monitor the temperature and will not flag any error on the temperature value.

17.4.1 Thermal Trip Pin

The PCH will provide a pin to the external world indicating that it has exceeded some temperature limit. The limit is set by BIOS. There is a limit register (PHL) which is compared to the present temperature. If the present temperature is greater than PHL then the pin is asserted (active low).

At boot, BIOS will program the limit and enable the sensor, which will allow the compare value to be sent to the pin. When used as PCH_HOT#, the pin is active low and the default state is that it is an O/D output with a pull-up on the board. Thus the pin is deasserted at boot or reset. BIOS sets PHL to enable the reporting.

The PCH will evaluate the temperature from the TS against the PHL every 1 second, so that the pin state can change at most once every 1 second. This prevents thrashing the state of the pin when the temperature is oscillating around the PHL value. There is no hysteresis on the value used.

17.5 Thermal Throttling

1. Additional programming to initialize Thermal Throttle device. These registers are platform specific and BIOS should follow the platform thermal management guide for the correct values.
2. If processor is capable of Cross Throttling and it is enabled
   a. Set TBAR + 40h [28:0] trip point temperature indicated by MSR 1A2h

      If (RATL mode) //refer to Skylake/Kaby Lake Processor BWG

      T0L = ((MSR 1A2[23:16] + 50) * 2)

      Else

      T0L = ((MSR 1A2[23:16] – MSR 1A2[29:24]) + 50) * 2

      T1L = T0L + 5

      T2L = T1L + 5

   b. Set TBAR + 40h [31:29] = 101b
   c. Set "PMC Throttling Enable", TBAR + 50h[14] = 1b
   d. Set PWRMBASE + 0C4[26:24] = 101b

   Else,

   a. Set T2L, TBAR + 40h [28:20] = 148h
   b. Set T1L, TBAR + 40h [18:10] = 142h
   c. Set T0L, TBAR + 40h [8:0] = 13Ch
   d. b. Set TBAR + 40h [31:29] = 101b
3. Per Interface Throttling Actions:

Per Interface Throttling Action		SKL PCH-H	SKL PCH-LP
DMI/OPI			
T0 Throttling Action	IOSF Private Cfg Space+2238h bit[11:10]	00	No throttling enabled on OPI
T1 Throttling Action	IOSF Private Cfg Space +2238h bit[9:8]	00	
T2 Throttling Action	0 IOSF Private Cfg Space +2238h bit[7:6]	01	
T3 Throttling Action	IOSF Private Cfg Space +2238h bit[5:4]	10	
SATA			
T1 Throttling Action - SSD	PCH-LP: SATA SIRI Index A0/A4/A8 [17:16] - Port 1 SATA SIRI Index A0/A4/A8 [1:0] - Port 0 PCH-H: SATA SIRI Index 0C/1C/2C [17:16] - Port 1 SATA SIRI Index 0C/1C/2C [1:0] - Port 0	01 - 1x Throttling	00 -no action for LPT-LP. PCH Cross Throttling Triggered first
T2 Throttling Action - SSD	PCH-LP: SATA SIRI Index A0/A4/A8 [19:18] - Port 1 SATA Index A0/A4/A8 [3:2] - Port 0 PCH-H: SATA SIRI Index 0C/1C/2C [19:18] - Port 1 SATA SIRI Index 0C/1C/2C [3:2] - Port 0	10 - 2x throttling	10 - 2x throttling
T3 Throttling Action - SSD	PCH-LP: SATA SIRI Index A0/A4/A8 [21:20] - Port 1 SATA SIRI Index A0/A4/A8 [5:4] - Port 0 PCH-H: SATA SIRI Index 0C/1C/2C [21:20] - Port 1 SATA SIRI Index 0C/1C/2C [5:4] - Port 0	11 - 4x throttling	11 - 4x throttling
Throttling configuration Port 1, 3, and 5 (SSD versus HDD)	PCH-LP: SATA SIRI Index A0/A4/A8 [27:24] PCH-H: SATA SIRI Index 0C/1C/2C [27:24]	These are 0000b for HDD, 1111b for SSD (Port 1)	

Per Interface Throttling Action		SKL PCH-H	SKL PCH-LP
Throttling configuration Port 0, 2, 4 (SSD versus HDD)	PCH-LP: SATA SIRI Index A0/A4/A8 [11:8] PCH-H: SATA SIRI Index 0C/1C/2C [11:8]	These are 0000b for HDD 1111b for SSD (Port 0)	
XHCI			
	xHCI MBAR+ 81B4h	[9:0] = 0h	[9:0] = 0h
	xHCI MBAR + 819Ch	[20] 1b, [16] 1b, [14] 1b, [13] 1b, [12] 1b, [11:8] 3h, [7:4] 7h, [3:0] Dh	[20] 1b, [16] 1b, [14] 1b, [13] 1b, [12] 1b, [11:8] 3h, [7:4] 7h, [3:0] Dh
Intel® ME		Defined in Intel ME Firmware	Defined in Intel ME FW

17.5.1 Memory Bandwidth Throttling

If the platform supports an external memory thermal sensor (TS-on-DIMM or TS-on-Board), system BIOS needs to program the registers below.

1. Program PWRMBASE + 0C8h[11:8] to the values associated with the GPIO pins connected to sensors on the platform which need to be communicated to the processor.

2. Pick two of 4 pins (recommend [11:10]) and route EXTTS#[1:0] to these two pins.

3. Set PWRMBASE + 0D4h [15:14] to 1 (assuming EXTTS#[1:0] are connecting to PWRMBASE + 0C8h [11:10]). This enable sending these two bits over PM_SYNC

4. Set PWRMBASE + 0F4h [15:14] to 1 (assuming EXTTS#[1:0] are connecting to PWRMBASE + 0C8h [11:10]). This enable sending these two bits over PM_SYNC in C0.

5. Set MCHBAR + 5880h [4] to enable EXTTS#. Pcode now will decode GPIO_C and GPIO_D from PM_SYNC message.

§ §

18 Integrated Clock Support

The PCH integrates the Integrated Clock Controller (ICC), which can provide better clock control than previous chipset generations and offers reduced BOM cost opportunities.

Please refer to the “Skylake/Kaby Lake Performance Tuning Guide” for more details on the Watchdog Timer and BCLK overclocking details.

Platforms that use only basic ICC configuration and don't allow any potentially unstable performance tuning operations may skip implementing WDT support described in the Guide. Instead, WDT must be disabled on these platforms. It is described in paragraph 18.5.1.

18.1 ICC Overview

The PCH operates in Full Clock Integration Mode, which fully utilizes the integrated clock controller to generate clocks for other system components.

Note: From Skylake PCH onwards, external clock mode is not supported by HW.

18.1.1 Full Clock Integration Mode Overview

The full clock integration mode utilizes the PCH internal clock controller to generate clocks for platform components. This mode takes inputs from an external crystal and generates the clock signals for other components on the platform. This mode only supports all ICC MEI messages. BIOS uses a combination of MEI controller and/or SBI accesses to configure the integrated clock. MEI interface has the most access, but is not available (or limited) prior to DRAM Init Done. SBI accesses can be used to configure registers that are only used by host and not required to be programmed out-of-band.

Overclocking is also supported in this mode for Skylake/Kaby Lake PCH-H.

18.2 BIOS Requirement

This section describes BIOS requirements for ICC during POST.

18.2.1 ICC MEI Messages

BIOS can control clock settings via the MEI messages during POST. The MEI #1 interface is used for ICC communication. All MEI messages should be sent after DID handshake. This section describes the ICC MEI messages overview and you can refer to Skylake/Kaby Lake Intel ME BWG for more MEI message details.

Table 18-1: ICC MEI Messages Availability

Command	Before EOP (BIOS)	After EOP (OS)
Get ICC Profile	Yes	Yes
Set ICC Profile	Yes	No
Get ICC Registers Lock	Yes	Yes
Set ICC Registers Lock	Yes	No
Get ICC Clocks Capabilities	Yes	Yes
Get Clock Range Definition Record	Yes	Yes
Get ICC Record	Yes	Yes
Set ICC Record	Yes	Yes
Direct Core Register Access*	Yes	No
Get HSIO Settings	Yes	Yes
Set HSIO Settings	Yes	No
Get Clock Settings	Yes	Yes
Set Clock Settings	Yes	Yes

18.2.1.1 ICC HECI Header

All ICC related HECI requests and responses use same common header. The data structure for that header is defined below.

Field	Length	Value	Description
API Version	4	00040000h	ICC API Version. 31:16-Major 15:0-Minor. SKLPCH-H/SKL PCH-LP – 00040000h
Command	4	Varies	Specifies the ICC HECI command to be performed and indicates the format of the request/response buffer following the header.
Status	4	Varies	On response, it indicates the status of the requested command. Should be 0 on request. See section below for specific ICC Status code definition. Note: For every response status that is not SUCCESS, command response ends after the ICC Header section
Length	4	Varies	Size in bytes of the data structure that follows the ICC Header.
Flags	4	00000000h	Bit0: 1-Posted Message. If set, no response is returned. Bit31-1: Reserved

18.2.1.2 ICC Command

The available ICC commands and their op-code values are listed below. Firmware should respond to unsupported commands with ICC_STATUS_INVALID_FUNCTION Status Code.

	Command	Description
0..15	Reserved	Reserved Command values. Used in previous generations, but not supported in this generation.
16	ICC_GET_PROFILE_CMD	Returns the information about the profile selected. This include the profile number, total profiles found. This information is also provided in the MBP data.
17	ICC_SET_PROFILE_CMD	This function allows BIOS to select the profile applied on boot. This value is saved within non-volatile memory.
18	ICC_GET_REGISTERS_LOCK_CMD	Function returns the registers to be unlocked after EOP. In general there are only three supported states (Default, All Locked, and All Unlocked).
19	ICC_SET_REGISTERS_LOCK_CMD	Prior to EOP, this function can be used by the BIOS to set the registers to be unlocked after EOP.
20	ICC_GET_CLOCK_CAPABILITIES_CMD	Returns information about the FW ICC capabilities such as SKU, HW versioning, and maximum record size.
21	ICC_GET_CLOCK_RANGE_DEF_REC_CMD	Returns the clock range definition record for the selected clock.
22	ICC_GET_RECORD_CMD	Returns the selected register record.
23	ICC_SET_RECORD_CMD	Validates and programs the specified register record.
24	ICC_GET_HSIO_SETTINGS_CMD	Returns the HSIO Survivability Settings record save in non-volatile memory
25	ICC_SET_HSIO_SETTINGS_CMD	Validates and stores the specified HSIO Survivability Settings record to non-volatile memory.
26	ICC_GET_CLOCK_SETTINGS_CMD	Returns the selected clock's frequency and spread settings
27	ICC_SET_CLOCK_SETTINGS_CMD	Sets the selected clock's frequency and spread settings.

18.2.1.3 ICC Status

The available return status codes are listed in table below.

	Status	Description
0	ICC_STATUS_SUCCESS	Command completed successfully.
1	ICC_STATUS_FAILURE	General internal failure occurred.
2	ICC_STATUS_INCORRECT_API_VERSI ON	Version in header not recognized by ICC FW
3	ICC_STATUS_INVALID_FUNCTION	Command specified in header not recognized.

	Status	Description
4	ICC_STATUS_INVALID_BUFFER_LENGTH	Length in header is out of range for the command specified
5	ICC_STATUS_INVALID_PARAMETERS	General parameter error indicating an unacceptable parameter specified in request of the ICC Header or request structure
6	ICC_STATUS_FLASH_WEAR_OUT_VIOLATION	Command attempted to write flash variables but was limited by the flash wear out protection. Retrying the command after reboot or time period, should allow write to occur.
7	ICC_STATUS_FLASH_CORRUPTION	Command attempted to access flash and the data was either corrupted, missing, or the flash management software indicated fatal failure.
8	ICC_STATUS_PROFILE_ NOT_SELECTABLE_BY_BIOS	Image configured to not allow runtime profile selection.
9	ICC_STATUS_TOO_LARGE_PROFILE_INDEX	Profile requested is beyond the max profile limit (15)
10	ICC_STATUS_NO_SUCH_PROFILE_IN_FLASH	Profile requested is not within the image or is zero sized.
11	ICC_STATUS_CMD_NOT_SUPPORTED_AFTER_END_OF_POST	Requested command is not supported after the end of POST message received. See table in “ICC HECI Commands Overview”.
12	ICC_STATUS_NO_SUCH_RECORD	The requested record does not exist.
13	Reserved	
14	Reserved	
15	ICC_STATUS_UOB_RECORD_TOO_LARGE	Specified Boot Settings is too large to be managed in the FW’s internal buffers. Record must be formatted into smaller configuration or FW buffers must be enlarged.
16	Reserved	
17	ICC_STATUS_REGISTER_IS_LOCKED	The accessed register is not allowed to be written. There is no expected usage for this register after EOP and is not expected to be written by OS software. Modifying lock with SetRegistersLocks command can change this behavior.
18	Reserved	
19	Reserved	
20	Reserved	
21	Reserved	
22	Reserved	
23	Reserved	
24	ICC_STATUS_UOB_RECORD_ IS_ALREADY_INVALID	Specified Boot Settings is attempting to invalidate an already invalid record (UOB).
25	Reserved	

	Status	Description
26	Reserved	
27	Reserved	
28	ICC_STATUS_RANGE_VIOLATION_ FREQ_TOO_HIGH	Request attempted to adjust the clock frequency beyond the frequency limits specified in the combined clock range definition.
29..35	Reserved	
36	ICC_STATUS_RANGE_VIOLATION_ FREQ_TOO_LOW	Request attempted to adjust the clock frequency below the frequency limits specified in the combined clock range definition
37..43	Reserved	
44	ICC_STATUS_SSC_MODE_CHANGE_ NOT_SUPPORTED	Request attempted to adjust the spread settings beyond the limits specified in the combined clock range definition
45..51	Reserved	
52	ICC_STATUS_NO_SUCH_TARGET_ID	Request attempted to access a target id not supported by the specified endpoint id.
53	ICC_STATUS_NO_SUCH_REGISTER	Request attempted to access a register not supported within the specified target id and endpoint id
54	Reserved	
55	ICC_STATUS_BUFFER_TOO_SMALL	Dynamic sized buffer is too small to return the requested registers.
56	Reserved	
57	ICC_STATUS_WAITING_FOR_POWER _CYCLE_RESET	Settings previously specified are pending. Further adjustments of settings are limited until pending settings applied. For example a profile change, limits any further boot settings from being applied.
58	ICC_STATUS_SURVIVABILITY_TABLE _ACCESS_VIOLATION	Specified HSIO Survivability settings violate the access permissions and are attempting to access HW outside the ModPHY.
59	ICC_STATUS_SURVIVABILITY_TABLE _TOO_LARGE	More HSIO Survivability settings specified than can be handled by the update mechanism.
60	ICC_STATUS_NO_SUCH_EID	Specified endpoint id does not exist in HW
61..	Reserved	
FFFFh	ICC_STATUS_UNSPECIFIED_ERROR	Internal unknown error status detected.

18.2.2 ICC MEI Messages Definition

This section provide definition for the ICC MEI command and response messages.

18.2.2.1 Get ICC Profile

Command used to get information about the profiles found within the image:

Information found in this command can also be found in the MBP data passed on initial boot. This is done to prevent BIOS delays as the CSME FW loads and is unavailable to respond to HECI requests.

Definition for GET_ICC_PROFILE command

Field	Length	Value	Description
ICC Header	20	Varies	ICC Command: GET_ICC_PROFILE
Parameters	0	NA	Command has no parameters

Definition for GET_ICC_PROFILE response

Field	Length	Value	Description	
ICC Header	20	Varies	ICC Command: GET_ICC_PROFILE	
NumICCProfiles	1	Varies	Number of motherboard profiles found in the image.	
			Bits 3:0	Number of profiles found.
			Bits 7:4	Reserved
OEM Profile Setup Parameters	1	Varies	Bits 3:0	Failsafe Profile Index
			Bits 6:4	Reserved
			Bit 7	Profile Is Selected By: Runtime Profile Selection Allowed Runtime Profile Selection Not Allowed
ICC Profile Index	1	Varies	Bits 3:0	Current Profile Selected
			Bits 7:4	Reserved
Reserved	1	0		
Profile Descriptions	N * 48		Array of strings (Pair per Profile) with following:	
Profile Name	24	Varies	Returns the profile name/description string associated with the selected profile.	
Profile Base	24	Varies	Returns the base profile string associated with the selected profile.	

18.2.2.2 Set ICC Profile

BIOS uses the Set ICC Profile HECI command to set the preferred profile index. This new profile is used only at next platform power cycle. CSME saves this profile in ICC Profile BIOS non-volatile memory.

This command is supported only during BIOS operation. Once the OS boots (after End Of POST message received) CSME stops support for this command.

Definition for SET_ICC_PROFILE command

Field	Length	Value	Description	
ICC Header	20	Varies	ICC Command: SET_ICC_PROFILE	
ICC Profile BIOS	1	Varies	Bits 3:0	BIOS selected profile for ICC FW to apply on next power-cycle.
			Bits 6:4	Reserved
			Bit 7	Revert to Failsafe – specifying 1, causes the BIOS selection to be ignored and the failsafe default is selected.
Reserved	3	0		

Definition for SET_ICC_PROFILE response

Field	Length	Value	Description
ICC Header	20	Varies	ICC Command: SET_ICC_PROFILE
Parameters	0	None	

18.2.2.3 Get ICC Registers Lock

The Get ICC Registers Lock command provides a mechanism to retrieve information about the ICC managed registers and interfaces available. The ICC HW has security registers that are used to limit register access to entities within the platform (Host, CSME). Key ICC registers used for ICC HW SKU's will always be locked out from direct access, but are accessible thru HECI commands. Other registers (like those used for CLKREQ) are meant to be configured by the BIOS. These can be accessed through the HECI interface or directly through the HW side band controller. At end of POST these interfaces are further limited.

Most ICC registers cannot be changed dynamically. ICC registers that can be changed dynamically must be done with deep understanding of ICC internals; otherwise, it can be harmful to platform stability. BIOS uses this command to limit ICC registers used by host after end of POST. Typically this would only be the small subset of registers for adjusting the BCLK and PCIe* clocks. These registers could also be locked for platforms not supporting overclocking/underclocking, but for simplicity it is recommended that they always remain unlocked. These registers will be limited by the CRDR for the profile, so locking them on certain SKUs just adds further burden.

If used before ICC application loads, the response to this command is delayed until ICC Application loads. To avoid delays, CSME populates the information found in this call within the MBP data available after the DID ACK message is received from CSME.

Definition for GET_ICC_REGISTERS_LOCK command

Field	Length	Value	Description
ICC Header	20	Varies	ICC Command: GET_ICC_REGISTERS_LOCK
Parameters	0	NA	None

Definition for GET_ICC_REGISTERS_LOCK response

Field	Length	Value	Description
ICC Header	20	Varies	ICC Command: GET_ICC_REGISTERS_LOCK
FW Lock State	1	Varies	0 – Default Lock 1 – All Locked 2 – All Unlocked Note: Registers limited by support of SKU
Lock Flags	1	Varies	Flags associated with the FW and HW locks Bit 0: 1: ICC HW Locks Enabled. Direct register interface not allowed. 0: ICC HW Locks Disabled and select registers can be accessed with HW mechanism. Bit 1: 1: ICC FW Locks Enabled. FW Lock State is applied through HECI functions and BIOS only APIs are disabled. 0: ICC FW unlocked Bits 7:4: Reserved
Reserved	2	0	

18.2.2.4 Set ICC Registers Lock

Prior to EOP, this function is available to BIOS to specify the FW Lock State. This Lock state specifies which registers will be available to the host OS. Typically this should be left to “Default Lock”. This leaves the registers associated with BCLK’s frequency and spread registers available for adaptive clocking and overclocking applications use. “All Locked” disables all writes to ICC registers through FW. This is available for platforms that don’t support any overclocking from the OS. “All Unlocked” leaves all pre-EOP registers available. This is meant only for debug.

Definition for SET_ICC_REGISTERS_LOCK command

Field	Length	Value	Description
ICC Header	20	Varies	ICC Command: SET_ICC_REGISTERS_LOCK
FW Lock State	1	Varies	0 – Default Lock 1 – All Locked 2 – All Unlocked Note: Registers limited by support of SKU
Reserved	3	0	

Definition for SET_ICC_REGISTERS_LOCK response

Field	Length	Value	Description
ICC Header	20	Varies	ICC Command: SET_ICC_REGISTERS_LOCK
Parameters	0	NA	

18.2.2.5 Get ICC Clock Capabilities

When overclocking is supported, it is required to deliver to BIOS ICC Control Library and to the OS application ICC Control Library some useful parameters per ICC platform settings. An initial list of these parameters appears in the command response below.

Definition for GET ICC CLOCKS CAPABILITIES command

Field	Length	Value	Description
ICC Header	20	Varies	ICC Command: GET_ICC_CLOCKS_CAPABILITIES
Parameters	0	None	

Definition for GET ICC CLOCKS CAPABILITIES response

Field	Length	Value	Description
ICC Header	20	Varies	ICC Command: GET_ICC_CLOCKS_CAPABILITIES
ICC HW Version	2	Varies	Helps host to identify ICC HW features per generation.
Reserved	2	0	Reserved for Future Use
ICC FW Version	4	Varies	Defines the FW's version/build number to host ICC applications.

Field	Bits	Description
Product Family	7..0	0-LPT-H 1-LPT-LP 2-WPT-LP 3-SKL PCH-H 4-SKL PCH-LP 6-SKL PCH-Server 7..255-Reserved
Stepping	15..8	0..9Fh-Reserved Axh- A0 Step A1h- A1 Step NNh- NN Step

Field	Bits	Description
Major	7..0	Matches FW Major Version #
Minor	11..8	Matches FW Minor Version #
Hotfix	15..12	Matches FW Hotfix #
Build	31..16	Matches FW Build #

Field	Length	Value	Description
Total NumClocks	1	2	Specifies the number of controllable clocks in the system. For SPT, this is 2 (BCLK and PCIe*). Value should not change based the clock is enabled/disabled. Changes based on the physical clock count.
Configurable Clocks	1	Varies	Specifies the enabled and configurable clocks in the system. This value varies based on the enabled system clocks.
Reserved	2	0	Reserved for Future Use
ICC Boot Status Events	4	Varies	ICC Boot Status Report. When bit set, it indicates event was seen. See table that follows.

Field	Bits	Description
IccBootRecoveryFailure	0	Boot Recovery Handler failed
IccWdtTimeoutRecovery	1	Detected OC WDT expiration
CMOSBatteryRemoval	2	Detected CMOS battery removal
RevertToProfile0	3	Detected failure caused profile to be revert to fail safes
ProfileSelectionFailure	4	ICC profile selection failed
ProfileIdxOutOfRange	5	Selected ICC profile index exceeds number of profiles in flash
BootSettingsInvalidated	6	UOB Record has been invalidated
OEMSettingsInvalid	7	ICC NVAR in flash is invalid or does not exist
CRDRCreateFailure	8	Combined Clock Range Definition Creation failed
CRDREnforcedToBasic	9	HW SKU limited the OEM's clock range definition
OEMMaxFreqViolation	10	OEM Max Frequency was limited based on HW SKU
OEMMinFreqViolation	11	OEM Min Frequency was limited based on HW SKU
OEMSpreadMaxViolation	12	OEM Max Spread Percentage was limited based on SKU
IntelCRDRViolation	13	Intel Record violated the Intel Clock Range Definition limits
IntelSettingsFailure	14	Intel Record not applied for reasons other than a CRDR violation.
OEMSettingsViolation	15	OEM Record violated the combined Clock Range Definition limits

Field	Length	Value	Description		
			OEMSettingsFailure	16	OEM Record not applied for reasons other than CRDR violation
			BootSettingsViolation	17	UOB Record violated combined CRDR limits
			BootSettingsFailure	18	UOB Record not applied for reasons other than combined CRDR violation
			Reserved	19	
			DynamicCtxApplied	20	Dynamic settings restored from Susram/flash backed record
			SetIccProfileReceived	21	FW received the SetIccProfile command
			Reserved	22	
			SetIccLockReceived	23	FW received the SetIccLock command
			SusramRecoveryFailure	24	Failed to restore Dynamic context from flash and susram state lost.
			PowerCycleResetPending	25	Further setting adjustments limited until power cycle reset occurs.
			WarmResetPending	26	Settings will/are pending to be applied when warm reset triggered. This flag is also reported in FWSTS register.
			Reserved	31..27	
ICC Flags	4	Varies	**Field**	**Bits**	**Description**
			Boot Settings Invalidation Reason	2..0	Invalidation reason associated with Boot Settings (bit6 in Boot Status)
			Reserved	3	
			ICC HW SKU	5..4	ICC HW SKU 0-Extereme 1-Medium (Reserved) 2-Basic 3-Enhanced
			Reserved	31..6	
ICC Debug Flags	32	Varies	Data reserved for ICC FW status. This could be a combination of status flags, state, or internal variables that varies between FW releases. It's expected that the Boot Recovery Status flags that contain details about UOB consumption, warm reset flow, invalidation reason, and internal state be returned in this data. Actual data format it left for definition by design.		

18.2.2.6 Get Clock Range Definition Record

Function retrieves the selected Clock Range Definition Record from the firmware.

Definition for GET CLOCK RANGE DEFINITION RECORD command

Field	Size (B)	Default	Description
ICC Header	20	Varies	ICC Command: GET CLOCK RANGE DEFINITION RECORD
Mode	1	Varies	0: Combined Clock Range Definition Record (calculated and used by the FW) 1: Intel Clock Range Definition Record (selected by FW per underlying ICC HW SKU). This mode is used for debug purposes. 2: Current selected OEM Clock Range Definition Record (selected by motherboard profile). This mode is used for debug purposes. 3: OEM Clock Range Definition Record to read is indexed by Required Record. This mode is used for debug purposes.
Required Record	1	NA	Useful only when Mode is 3. This parameter is the index of OEM profile to read.
Reserved	2	NA	Reserved – MBZ

Definition for GET CLOCK RANGE DEFINITION RECORD response

Field	Size (B)	Default	Description
ICC Header	20	NA	ICC Command: GET CLOCK RANGE DEFINITION RECORD
CRDR	16	NA	Requested Clock Range Definition record. Only records associated with BCLK and PCIe* are returned.

18.2.2.7 Get ICC Record

Function retrieves the selected ICC Record from the firmware.

Definition for GET ICC RECORD command

Field	Size (B)	Default	Description
ICC Header	20	Varies	ICC Command: GET ICC RECORD

Field	Size (B)	Default	Description
Requested Record	1	Varies	Species the record to be retrieved 0 – Intel Record 1 – UOB Record 2 – Pre-UOB Record 3 – Current Record – Return the registers based on ICC Record that follows 4 – Current ICC Registers – Return the registers based on an internal selection. See CSME_ICC_REGS_SPT.xlsx for list of registers. Format of the record determined by FW. 19-5 – Reserved 20 – OEM Record Profile 0. Note this is only the OEM record data and not the CRDR. To retrieve the CRDR use GetClockRangeDefinitionRecord. 21 – OEM Record Profile 1 22 – OEM Record Profile 2 23 – OEM Record Profile 3 24 – OEM Record Profile 4 25 – OEM Record Profile 5 26 – OEM Record Profile 6 27 – OEM Record Profile 7 28 – OEM Record Profile 8 29 – OEM Record Profile 9 30 – OEM Record Profile 10 31 – OEM Record Profile 11 32 – OEM Record Profile 12 33 – OEM Record Profile 13 34 – OEM Record Profile 14 35 – OEM Record Profile 15 255..36 – Reserved
Reserved	3	NA	Reserved – MBZ
ICC Record	Variable	Varies	When requested record is CurrentRecord(3), this specifies the ICC record that will be parsed and populated with the current HW register values. The record's registers section should have initialized flags, bundle counts and ICC Address Masks. Values in the record will be overwritten in the response. For other requested record types, this part of the request should be 0 bytes in length.

Definition for GET ICC RECORD response

Field	Size (B)	Default	Description
ICC Header	20	Varies	ICC Command: GET ICC RECORD
Reserved	4	0	Reserved. Used to maintain symmetry in the command/response buffers.
ICC Record	N	Varies	Requested record. Variable length.

When requesting the UOB record, if no UOB record exists or it is zero sized, the FW returns a successful status and an empty record (no registers section) with valid bit cleared and Invalidate Reason set to 000 - Power Loss in the record flags. When requesting a UOB record that was invalidated, the FW returns a successful status and the invalid record found.

18.2.2.8 Set ICC Record

This command performs the low level write to ICC registers using the specified record. Prior to performing any write, the record is first processed to validate that the values specified are allowed based on the platform's SKU and profile selection.

Definition for SET_ICC_RECORD

Field	Size (B)	Default	Description
ICC Header	20	Varies	ICC Command: SET_ICC_RECORD
Requested Record	1	Varies	Species the record to be retrieved

0	Intel Record – Reserved
1	UOB Record
2	Pre-UOB Record – Reserved
3	Current Record – Write Registers
19-4	Reserved
20	OEM Record Profile 0 - Reserved
21	OEM Record Profile 1 - Reserved
22	OEM Record Profile 2 – Reserved
23	OEM Record Profile 3 – Reserved
24	OEM Record Profile 4 – Reserved
25	OEM Record Profile 5 – Reserved
26	OEM Record Profile 6 – Reserved
27	OEM Record Profile 7 – Reserved
28	OEM Record Profile 8 – Reserved
29	OEM Record Profile 9 – Reserved
30	OEM Record Profile 10 – Reserved
31	OEM Record Profile 11 – Reserved

Field	Size (B)	Default	Description
			32 – OEM Record Profile 12 – Reserved 33 – OEM Record Profile 13 – Reserved 34 – OEM Record Profile 14 – Reserved 35 – OEM Record Profile 15 – Reserved
Reserved	3	NA	Reserved – MBZ
ICC Record	Variable	Varies	Record must have header flags and RecordLength initialized. When updating UOB, flags "UpdateOnWarmReset", "Valid", and "InvalidateReason" should be specified. Other flags may be ignored. The bundles in the section are processed in specified order and written to the HW. Prior to writing any value in the record, all registers are tested to be within acceptable ranges. On failure, no register values are written.

Definition for SET_ICC_REGISTER response

Field	Size(B)	Default	Description
ICC Header	20	NA	ICC Command: SET_ICC_REGISTER
Bytes Processed	2	0	Specifies where within the record processing stopped. On success, the caller can expect that Bytes Processed to equal the record length specified in the record header. On failure, this indicates the offset within the record where an error was detected and processing stopped. This should reference a value specified in the registers section.
Reserved	2	0	Reserved to align structure size to 4-byte boundary.

CSME FW verifies each register delivered through the ICC Record are not locked and that its values do not violate Combined Clock Range Definition Record rules. FW uses the algorithm implemented in ICCRecordWrite function to first verify the registers without committing them to H/W. Once verified, a second pass with write flag is used to write the values to HW.

Write UOB has been implemented accordingly with table below. FW reads existing UOB from Flash only for invalidation. FW writes the new UOB directly without an extra invalidation step. When writing the UOB record, additional state is maintained within the record flags. The following table describes how the state of the UOB is check.

UOB State (In Write)	UOB State (In Flash)	Description
VALID	VALID	SUCCESS – Overwrites the existing UOB.Done this way to prevent need for 2 UOB writes (1 to invalidate and 1 to write new value).
INVALID	VALID	SUCCESS – Invalidates the existing UOB with specified invalidation reason.
VALID	INVALID	SUCCESS – Overwrites the existing UOB. Invalidation reason can be maintained or set to 0. Technically a don't care with valid flag is cleared.
INVALID	INVALID	ICC_STATUS_UOB_RECORD_IS_ALREADY_INVALID – Write prevented, so that invalidation status is maintained.

In CSME FW, all variables should be created by default within the build process. FW should not ever expect a case where the UOB in flash can't be found.

UOB Consumed flag in the FW context should be cleared after writing the UOB. In addition dynamic settings should not be saved in dynamic context until the UOB is consumed. This is necessary to avoid issues where UOB settings are used to ramp the clock.

Prior to creating the UOB, the pre-UOB record should have been created. FW may want to debug assert if this condition does not exist. PreUOB values expected to be read before UOB and saved later to flash.

18.2.2.9 Get Clock Settings

Function retrieves the selected clock's current settings.

This NEW function is provided to reduce the ICC library's complexity and provide a function that abstracts the ICC HW's implementation and exposes the supported clocking capabilities.

See 18.2.2.11 Get/Set Clock Setting Request/Response Structure for details on the request/response structure. This command uses a common structure format. Reserved fields ignored.

Command	Minimum Request Buffer	Minimum Response Buffer
GET_CLOCK_SETTINGS	24	48

18.2.2.10 Set Clock Settings

Function sets the selected clock's frequency and spread settings.

This NEW function is provided to reduce the ICC library's complexity and provide a function that abstracts the ICC HW's implementation and exposes the supported clocking capabilities.

Note: The clock settings cannot be adjusted while there is a profile change pending.

See 18.2.2.11 for details on the request/response structure. This command uses a common structure format. Reserved fields ignored.

Command	Minimum Request Buffer	Minimum Response Buffer
SET_CLOCK_SETTINGS	48	28

The following table specifies how the FW should use the combination of flags to create and apply the appropriate records.

Applied On Warm Reset	Type	Set To Default	Description
Don't Care	Dynamic	1	The pre-UOB is used to attempt to ramp to the target settings back to their default settings.

Applied On Warm Reset	Type	Set To Default	Description
Don't Care	Permanent	1	FW invalidates the UOB with status indicating user requested invalidation. In addition it returns status indicating that a power cycle reset is pending.
0	Dynamic	0	FW creates a dynamic record to achieve the target frequency and spread settings, and then applies the settings using the clock ramping algorithm.
0	Permanent	0	FW creates a UOB record with the target frequency and spread settings. The settings are updated into a SusRam and Flash variables and returns a status indicating that power cycle reset pending.
1	Dynamic	0	FW creates a dynamic record to achieve the target frequency and spread settings using the CSME DMI PLL CONFIG register and applies them in HW. This is only applicable for BCLK. If specified with PCI clock an error should be returned. Once applied a status indicating a warm reset pending is returned.
1	Permanent	0	FW creates a UOB record to achieve the target frequency and spread settings using the CSME DMI PLL CONFIG register. This is only applicable for BCLK. If specified with PCI clock an error should be returned. Once applied a status indicating a power cycle reset pending is returned.

Note: When type "Both", FW should first perform the Permanent operation then the dynamic application of the record.

Note: Once the permanent UOB has been applied, the FW should not save/restore the dynamic record until the new UOB has been consumed.

18.2.2.11 Get/Set Clock Setting Request/Response Structure

The following defines a common request/response buffer format for the GetClockSettings and SetClockSettings HECI commands

Field	Size (B)	GC RQ	GC RP	SC RQ	SC RP	Description
ICC Header	20	X	X	X	X	ICC Command: GET_CLOCK_SETTINGS or SET_CLOCK_SETTINGS
Requested Clock	1	X		X		Specifies an index of the clock capabilities to "Get" or "Set". This index should be less than 2 (BCLK and PCIE clocks)
Setting Type	1	X				Selects the source of to retrieve the clock settings. If not found it the selected settings, defaults to current settings. **Value** / **Description** 0 – Current Settings – Returns the clock frequency/spread based on the currently active HW register values.

Field	Size (B)	GC RQ	GC RP	SC RQ	SC RP	Description
						1 – Boot Settings – Returns the clock frequency/spread based on the pre-UOB and UOB settings. 2 – Pre-Boot Settings – Returns the clock frequency/spread based on the pre-UOB settings only 3 – OEM Boot Settings – Returns the clock frequency/spread based on the OEM and HW boot settings. 4..255 – Reserved.
Flags	1			X		Flags used to determine how FW should interpret the requested fields. **Bit(s)** / **Field** / **Description** 0 / Round Down / Specifies if FW should round up (0) or round down (1) the specified User Frequency when converting it to a divider 1 / Translate Freq. Only / When set, indicates to FW that Set Clock Settings should translate the User Frequency and return actual frequency. No write is actually performed 7..1 / Reserved / Reserved
Reserved	1					Reserved
Frequency	4		X		X	Frequency of the clock in Hz, based on the current divider settings
User Frequency	4		X	X		User defined frequency. HW frequency is actually determined using a set of dividers, meaning exact user frequency cannot always be achieved. To manage this, FW tracks the user requested frequency.
Max Frequency	4		X			Specifies the maximum frequency supported by the clock. This is determined from the combined CRDR
Min Frequency	4		X			Specifies the minimum frequency supported by the clock. This is determined from the combined CRDR
SSC Mode	1		X	X		Spread Spectrum Clocking Mode (Up, Down, Center, None), based on the HW register settings. Only Down-spread and None currently supported.
SSC Percentage	1		X	X		1/100 percentage of the frequency range to spread. Example: a value of 50 indicates 0.50%. Value based on user requested setting.
Max SSC Percentage	1		X			1/100 percentage of the frequency range to spread. Example: a value of 50 indicates 0.50%. Value specifies max spread percentage that can be applied to the clock. In addition it is limited based on the current frequency and the frequency limits specified in the CRDR. Exception is when min frequency equals max frequency. In this case, spread can be enabled /disabled from max % to 0.

Field	Size (B)	GC RQ	GC RP	SC RQ	SC RP	Description
Reserved	1					Reserved.
Current Flags	2		X	X		Flags indicating current state of clock

Bit(s)	Field	Description
1..0	Ratio	If supported, defines the ratio to be applied with the clock. 0-5/5, 1-5/4, 2-5/3, 3-5/2
2	Reserved	Reserved for future use.
3	Applied On Warm Reset	When set, this indicates that the clock settings were applied on warm reset.
5..4	Type	Specifies how the settings with associated clock were applied. 0 – Permanent indicates UOB used 1 – Dynamic indicates no UOB and not default 2 – Both indicates UOB used and dynamic 3 – Reserved
6	Set To Default	When set, the clock settings are at their default settings. UOB invalid and current settings equal to pre-UOB settings.
7	Disabled	When set, the clock is in a disabled state and is not in use. Not supported at runtime.
8	Dynamic Settings Applied	When set, this indicates that runtime changes have been applied. preUOB/UOB settings not equal to current register settings. Reserved in Set Clock Settings.
9	WarmResetPending	Settings applied on warm reset pending. Flag set when “AppliedOnWarmReset” flag set and cleared once warm reset occurs. Reserved un Set Clock Settings.
10	PowerCyclePending	New Boot settings applied and power cycle reset pending. Flag cleared once boot settings are consumed. Reserved in Set Clock Settings.
15..11	Reserved	Reserved for future use.

Support Flags			X			Flags indicating the supported clock setting changes. Determined from combined CRDR.

Bit(s)	Field	Description

Field	Size (B)	GC RQ	GC RP	SC RQ	SC RP	Description		
						0	Ratio	Ratio selection allowed
						1	DownSpread	Down spread SSC supported
						2	UpSpread	Up spread SSC supported
						3	CenterSpread	Center spread SSC supported
						4	ChangeAllowed	Frequency and Spread changes allowed
						5	HaltAllowed	Spread can be enabled or disabled
						6	PCIe*	Clock uses the PCIe* clock source
						7	BCLK	Clock uses the BCLK clock source
						15..8	Reserved	Reserved

Note: The settings in grey are reserved fields when the structure used in the specified command. Green fields with 'X' are expected to be populated with valid values.

GCRQ – Get Clock Settings Request

GCRP – Get Clock Settings Response

SCRQ – Set Clock Settings Request

SCRP – Set Clock Settings Response

18.2.2.12 Get HSIO Settings

Function used to retrieve the current HSIO Survivability settings that are provided to the PMC for application. This interface may be disabled when FW receives the EOP message.

Definition for GET_HSIO_SETTINGS command

Field	Size (B)	Default	Description
ICC Header	20	Varies	ICC Command: GET_HSIO_SETTINGS
Reserved	4	NA	Reserved – MBZ

Definition for GET_HSIO_SETTINGS response

Field	Size (B)	Default	Description
ICC Header	20	Varies	ICC Command: GET_HSIO_SETTINGS
Reserved	4	0	Reserved. Used to maintain symmetry in the command/response buffers.
HSIO Survivability Table	MAX	Varies	Specifies the HSIO Survivability table. The first DWORD of this structure is passed to BIOS in the pre-DID messaging to identify the HSIO Survivability settings.

Field	Size (B)	Default	Description
CRC	2		CRC calculated across the HSIO entries.
Version	1		Version, chipset, and stepping this table targets
Product	1		
GroupTable[16]	32	Varies	Table used to define groups of registers. Contains group ID and offset into entry table.
Entries[]	Varies	Varies	Each entry defines an IOSF SB address and value.

18.2.2.13 Set HSIO Settings

Function used to program the HSIO Survivability. This table is applied on the next global reset.

Definition for SET_HSIO_SETTINGS command

Field	Size (B)	Default	Description
ICC Header	20	Varies	ICC Command: SET_HSIO_SETTINGS
Reserved	4	NA	Reserved – MBZ
HSIO Survivability Table	MAX	Varies	Specifies the HSIO Survivability table. The first DWORD of this structure is passed to BIOS in the pre-DID messaging to identify the HSIO Survivability settings
CRC	2		CRC calculated across the HSIO entries.
Version	1		Version, chipset, and stepping this table targets
Product	1		Bits: Description 7..4: Product Family 0-LPTH. 1-LPTLP, 2-WPTLP, 3-WBG, 4:7-Reserved, 8-SKL PCH-LP, 9-SKL PCH-S, 10-LBG, 11:15-Reserved 3..0: Stepping
GroupTable[16]	32	Varies	Table used to define groups of registers. Contains group ID and offset into entry table.
Entries[]	Varies	Varies	Each entry defines an IOSF SB address and value.

Definition for SET_HSIO_SETTINGS response

Field	Size (B)	Default	Description
ICC Header	20	Varies	ICC Command: SET_HSIO_SETTINGS
Parameters	0	NA	None

18.2.3 MEI MKHI Messages

These MKHI messages are used for communicating ICC requests and responses prior DID handshake.

Table 18-2: MKHI MEI Messages Availability

Command	Prior DID (BIOS)	After EOP (OS)
ICC_CMD_REQ	Yes	No
ICC_CMD_RESP	Yes	No

18.2.3.1 Intel ME Kernel Hos Interface (MKHI) MEI Header

All MKHI related MEI requests and responses define the Intel® MEI messages for host and Intel ME Kernel communication. These messages are used to communicate prior to memory initialization. After the DID message is received, all further communication must be done through the normal HECI messaging interfaces. The data structure for that header is defined below. For more information on this messages please refer to Skylake/Kaby Lake Intel ME 11 BIOS Specification.

Table 18-3: Definition of MKHI MEI Header

Field	Length (bit)	Value	Description
Group Id	8	F1h	Kernel Client specific information for dispatching
Command	7	N/A	Command code specific to MEI client ICC_CMD_REQ = 01h
IsResponse	1	N/A	Set to 1 in response message
Reserved	8	00h	Reserved
Result	8	N/A	Result of the request message

18.2.3.2 ICC Command Request (ICC_CMD_REQ)

After receiving this message, the FW is expected to return an acknowledge message back to the BIOS.

The ICC request/response flow is used when doing the warm reset BCLK overclocking flow. If you are performing BCLK dynamically, there is no need to send this message. Sending this message will flag the ICC fw that a BCLK ramp needs to occur. This flow is documented in detail in the Skylake/Kaby Lake Processor Performance Tuning Guide.

Table 18-4: Definitions for ICC Command Request Message

Field	Length (byte)	Value	Description
MEI Header	4	80080008h	Refer to Skylake/Kaby Lake ME11 BIOs specification for more details.
MKHI Header	4	000001F1h	Command code specific to MEI client

Field	Length (byte)	Value	Description
			ICC_CMD_REQ = 01h
ICC Command Request Data	4	N/A	Refer to Table 18-5 for definitions on ICC Command Request Data

Table 18-5: Definitions for ICC Command Request Data

Field	Length (bits)	Value	Description
BClkRampOnWarmResetEn	0	N/A	Setting this bit, causes the Intel ME FW to perform the warm reset flow configured on a previous boot. Clearing this bit or not sending this command causes the BCLK ramp on Warm reset to be skipped.
Reserved	24..1	000000h	Reserved
ICC Message Type	31..25	6h	A value of 6 indicates this is an ICC command type.

18.2.3.3 ICC Command Response (ICC_CMD_RESP)

This command is used by Intel ME to acknowledge the BIOS message in bits 25-31 refer to Table 18-7 for details.

Table 18-6: Definitions for ICC Command Response Message

Field	Length (byte)	Value	Description
MEI Header	4	80088008h	Refer to Skylake ME11 BIOs specification for more details.
MKHI Header	4	000081F1h	Command code specific to MEI client ICC_CMD_RES = 81h
ICC Command Response Data	4	N/A	ME used this message to acknowledge the BIOS message. Please refer to Table 18-7 for definitions on ICC Command Response Data

Table 18-7: Definition of ICC Command Response Data

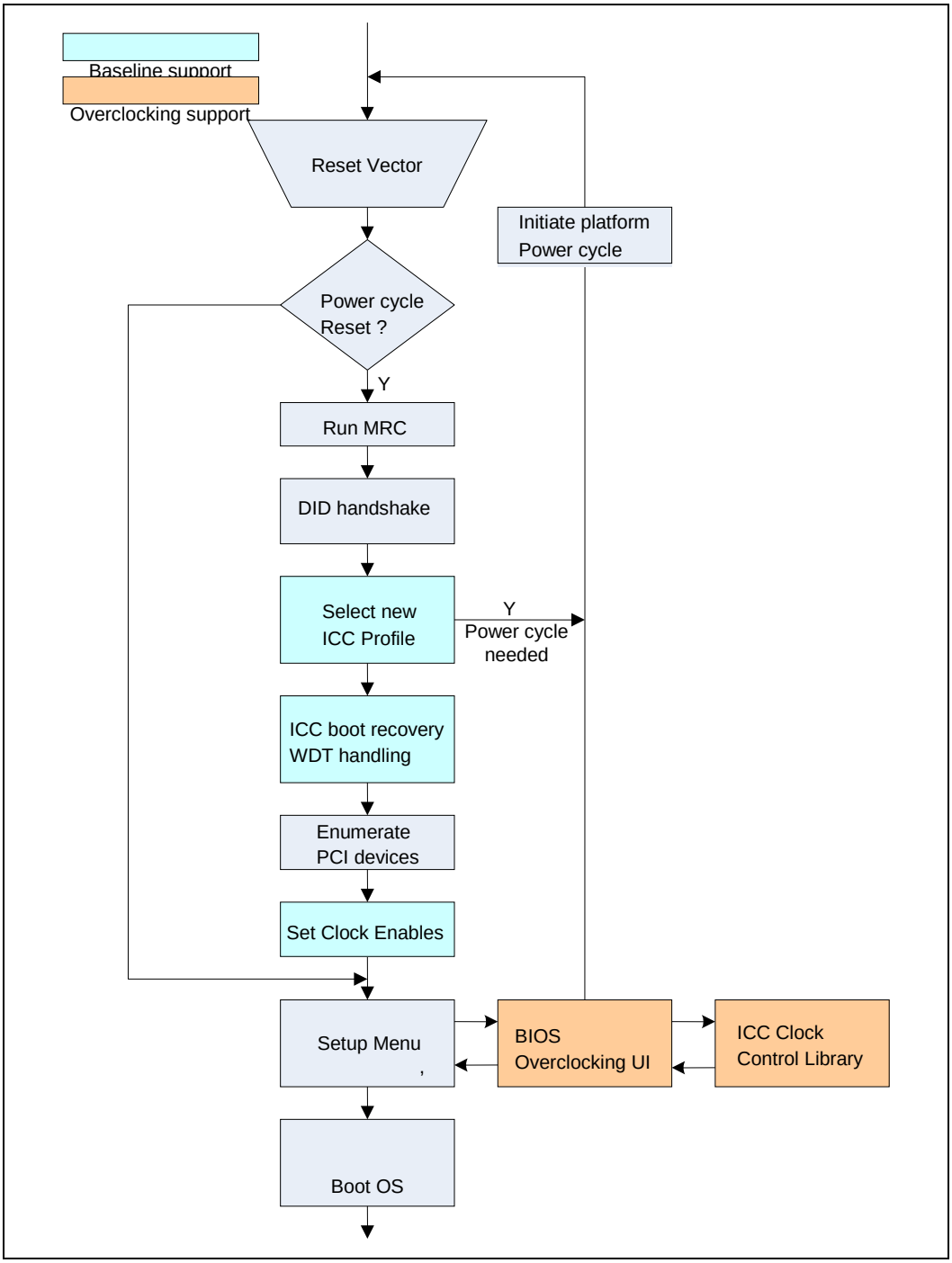
Field	Length (bits)	Value	Description
FWSTATUS	24..0	N/A	These bits define Intel ME operational state. Please refer to Skylake ME11 BIOS Specification for more details.
Acknowledgment Data	27..25	0h	This reports the status of the request. For ICC message requests a value of 0 is always returned.

BIOS ICC message Acknowledge	31..28	6h	For ICC message acknowledgment this field is populated with 6. Once populated, the ACK data is considered valid. After sending the message request, the acknowledgment indicates request completion

18.2.4 Full Clock Integration Mode

There are 2 levels of BIOS support for FCIM, baseline and extended ICC configuration. The baseline ICC configuration is required as minimum BIOS support for all platforms. The extended ICC configuration is supported through the Get/Set Clock Settings MEI messages to support overclocking/underclocking on high performance systems or some specialized platforms. All MEI messages are supported in this mode.

Figure 18-1: BIOS POST Flow



18.2.4.1 Baseline ICC Configuration

The BIOS requirements for the Baseline ICC Configuration are listed below:

1. ICC Profile selection and verification in early POST.
2. Gate unused clocks to save power after PCI enumeration.
3. ICC settings are automatically locked down at EOP.

18.2.4.2 Extended ICC Configuration

The extended ICC configuration is provided through the Get/Set Clock Settings MEI messages. The extended ICC configuration is useful for gaming and other performance oriented applications, as well as lower clock frequencies for specific hardware requirements like mitigating EMI among onboard communication devices. The BIOS requirements of the extended ICC Configuration are listed below. Refer to the Skylake/Kaby Lake Platform Performance Tuning Guide for more details.

1. Modify clock value within the pre-defined range for each clock separately, i.e. BCLK, DMI, PCIe*, PCI, USB, IGD, etc.
   a. Only 1-2 adjustable clocks (varies based on configuration). If in standard configuration, only modifications to the PCIe* clock is allowed and is very limited (100MHz 0.5% downspread with option to enable/disable spread). PCIe* clock is used to generate the DMI, BCLK, DP, PCIe*, SATA, and USB 3.0 clocks. In other configurations, a second clock source (BCLK) is used to source DMI/BCLK. Details of the clock configurability is based on the SKU'g and image configuration.
2. BIOS needs to provide WDT support for OS drivers and applications.
3. When the WDT timer expired, Intel ME FW returns to the fail-safe settings configured in the image by invalidating any settings provided from host and recover from the failure. BIOS should check WDT status in early POST and execute in recovery path if needed when ICC setup or OS applications caused system hang and WDT expired.

18.2.4.3 Other Considerations

1. The Set Icc Registers Lock Intel® MEI message controls the availability of icc registers in the OS after EOP. Typically, this should be left to “Default Lock”. This leaves the registers associated with BCLK’s frequency and spread registers available for adaptive clocking and overclocking applications use. In this case alternatively, OEMs may choose not send Set Icc Registers Lock Intel® MEI message, and allow the Intel ME FW to apply the default lock to ICC registers.
2. If OEM does not use WiMAX or Intel® Integrated Graphics and if the platform does not support BCLK overclocking then the recommended lock state would be “All Locked” which will disable all writes to ICC registers through FW.
3. “All Unlocked” leaves all pre-EOP registers available. This is meant only for debug.

18.3 Clock Gating

At platform initial power up, all clock outputs are enabled by default, unless configured differently in the image using the Intel ME image configuration tools. It is

recommended that the BIOS perform the clock gating at the end of PCI enumeration to disable clocks to unused PCI / PCIe* slots or during hot plug events.

In addition to clock gating, BIOS should also perform L1 substate initialization and enable HW PM if supported by the device. Refer to the PCIe* specification for details on L1 substate and HW PM programming.

18.4 ICC Profile Selection and Verification

In Skylake/Kaby Lake PCH ICC implementation, ICC can supports up to 16 profiles on a single SPI image. Profiles provide a mechanism to configure ICC settings during BIOS image creation using Intel ME provided image tools. Typical configurations are used to adjust the "Standard" configuration to configurations for "Overclocking" and/or "Adaptive-Clocking". In addition, a BIOS may have configurations with platform specific settings like power-on CLKREQ routing/enables, power-on clock enables, power management related settings, and/or power-on clock frequency/spread settings. BIOS can specify the default fail-safe profile at image creation, and if enabled can perform a runtime selection using Set_ICC_Profile MEI message.

Note: On Skylake/Kaby Lake PCH, the ICC straps have been removed.

Note: This assumes the failsafe profile is adequate enough to get host running to allow Intel ME to become fully functional (typically DMI and DRAM memory initialization). In addition, optimal defaults are specified in the tools based on reference platform settings and not expected to vary except where there are known exceptions

BIOS can perform ICC Profile selection steps below:

1. BIOS checks which board/platform profile will be selected by using predefined board/platform SKU detection mechanisms.
2. BIOS reads current ICC profile from Intel ME by
   a. Retrieving it through Intel ME to BIOS Payload (MBP) data (preferred)
   b. Sending Get_ICC_Profile MEI message.
3. If the image contains multiple profiles and the profile selection is enabled, BIOS should display the current profile and allow user selection of alternate profiles.
4. If the selected profile is not the same as current profile, BIOS can send the new profile selection to Intel ME though Set_ICC_Profile MEI message and perform a global reset afterward.

Note: Settings are not applied until a host-power cycle reset is performed.

18.5 Watchdog Support

In PCH, ICC uses a watchdog timer, OC_WDT. BIOS contains a WDT driver that allows other modules to use the WDT without knowing details of its operation. ICC needs to use the WDT as a mechanism to recover itself from platform instability caused by over clocking, and also for detecting spurious reset.

Note: The same WDT can also be used by the processor host for failsafe purpose.

18.5.1 WDT Support for Basic ICC configuration

On platforms that support Basic ICC configuration only and don't allow any kind of potentially unstable performance tuning, it is allowed to not implement full WDT support, as described in "Skylake/Kaby Lake Performance Tuning Guide". Instead, WDT hardware must be disabled.

WDT could be disabled by platform policy. To disable WDT, make sure WDT is not used by any module and disable WDT by policy.

18.5.2 WDT Support for Extended ICC Configuration

BIOS should use the WDT driver to activate the WDT before implementing potentially dangerous ICC changes. The WDT will be deactivated by WDT driver when platform boots to operating system ACPI Support for WDT

The WDT registers are accessed through existing LPC register space and do not have its own PCI device config space. In order to load the WDT driver, BIOS needs to define the WDT as ACPI device. To allow loading of the WDT driver when needed and avoid yellow bang at Windows* boot, BIOS needs to define _CID object for the WDT.

18.6 ICC Control Library on Skylake / Kaby Lake

Extended ICC Configuration is available through the GetSet Clock Settings MEI messages. The ICC Control Library is removed on Skylake/Kaby Lake. The ICC proxy service still exists for OS usage, but the BIOS ICC library has been replaced with direct MEI messaging of the clock configuration. Please refer to ICC SDK manual and Performance Tuning Guide for details.

§ §

19 Power Management

The chapter listed out some of the power management concerns that BIOS should be consider.

19.1 Initialization

System BIOS must allocate and assign base address to ABASE and PWRMBASE and enable them accordingly. Due to the DMI PMC source decode registers limitation, the PWRMBASE is to be allocated a 64KB address space (even if it is 4KB in size). Refer to section 5.9.1 for the fixed address allocated for PWRMBASE.

19.2 Handling Status Registers

System BIOS must set '1b' to clear the following registers during power-on and resuming from Sx sleep state.

- PWRMBASE + 10h[0] = 1b
- PWRMBASE + 10h[4] = 1b, needs to be done as early as possible during PEI
- PWRMBASE + 10h[5] = 1b

19.3 Power Button Override and AfterG3_EN Bit Usage

The PCH Power Button implementation is described below:

1. The power button override (4 seconds continuous pushing) will cause the PCH to transition to the S5 state EVEN if it is already in states S3-S4.
   a. The override period is configurable through "Power Button Override Period", PWRMBASE + 3Ch bit [31:29].
   b. The PMC will continue to observe the power button signal even if the GPD_3 is configured as GPI and not native as long as the "PB_DIS", PWRMBASE + 3Ch bit [28] is not set.
2. When a Power Button Override event occurs, except those specified in PCH EDS "Causes of Wake Events" with Power Button Override. Those wake events cannot be disabled.
3. When a PWRBTN# override event occurs, the wake events will also be disabled (except for PWRBTN#), which cannot be disabled as a wake event.
4. The System BIOS can set the AFTERG3_EN bit before entering the system states (S3-S5), so in case of a power failure and restoration of power back into the system, the system does not power up automatically.
5. System BIOS can read the current state of the power button in PMC PCI offset A0h[9]. System BIOS can wait until the power button is released before transitioning into a suspend state in order to ensure that the user is not generating a power button override.
6. AFTERG3_EN bit may be set/cleared by certain hardware events, please refer to PCH EDS for more detail. In such circumstances, actual AFTERG3_EN might not be in sync with policy set in BIOS setup option. The recommendation is BIOS should always update the AFTERG3_EN bit according to setup option.

19.3.1 Intel® ME Enabled System after G3 Exit Behavior

On Intel ME enabled systems getting power back from G3 may have different power state transitions based on AfterG3_En bit in PCH and Intel® ME Power Package selection. If the Intel® ME Power Package selected is PP1, Intel® ME will remain off after returning from a G3 state. If the Intel® ME power package option setting selected is PP2, Intel® ME will wake up to M3. As for the combination of AfterG3_En bit and Intel® ME power package selection, refer to Table 19-1.

Table 19-1: Intel® ME Enabled System Behavior After G3 Exit

	General PM Configuration 3 A4h [bit 0]	Intel® ME Power Package Selection	Proper ACTION
A	1 (go to S5)	PP2	Intel ME will turn on in M3 state
B	1 (go to S5)	PP1	Intel ME and Host will remain off
C	0 (go to S0)	PP2	Intel ME and Host will turn on and boot to M0/S0
D	0 (go to S0)	PP1	Intel ME and Host will turn on and boot to M0/S0

Note: Refer to MEBX user guide for details on available Intel® ME power package selections.

19.4 Power Failure Considerations

There are several bits in PCH that indicate the power failure conditions.

PWR_FLR bit, GEN_PMCON_3 (PMC PCI offset A4h[1])

When the PWR_FLR bit is set, it indicates the trickle power from the main battery or trickle supply failed while in suspend or since last boot. This bit is in the RTC Well and is cleared only by RTCRST#. Software writes a “1” to clear this bit. System BIOS should follow cold boot path if PWR_FLR, GEN_RST_STS or PWRBTNOR_STS is set to 1 regardless of the value in the SLP_TYP field.

RTC_PWR_STS bit, GEN_PMCON_B (PMC PCI offset A4h[2])

When the RTC_PWR_STS bit is set, it indicates that the RTCRST# signal went low. Software should clear this bit. For example, changing the RTC battery sets this bit. System BIOS should reset CMOS to default values if the RTC_PWR_STS bit is set.

The System BIOS should execute the sequence below if the RTC_PWR_STS bit is set before memory initialization. This will ensure that the RTC state machine has been initialized.

1. If the RTC_PWR_STS bit is set which indicates a new coin-cell battery insertion or a battery failure, steps 2 through 5 should be executed.
2. Set RTC Register 0Ah[6:4] to 110b or 111b
3. Set RTC Register 0Bh[7].
4. Set RTC Register 0Ah[6:4] to 010b
5. Clear RTC Register 0Bh[7].

19.5 Additional Power Management Programming

System BIOS requires to program the registers listed below for platform power management.

- Set PMC PCI offset A0h [4] after GBL_SMI_EN has been programed. PMC PCI offset A0h [30:27, 23, 10, 1, 0] are platform specific.
- Set PMC PCI offset A8h [14, 10, 9] to all 1. This register MUST be programmed before PM_SYNC_MODE is programmed.
- Set PMC PCI offset ACh [31] to 1b.
- Set PWRMBASE + offset 314h = 1E0A4616h.
- Set PWRMBASE + offset 31Ch [8:0] = 003h
- Set PWRMBASE + offset 31Ch [29,27,25,24,23] = each to 1b
- Set PWRMBASE + offset 31Ch [22,15,13] = each to 0b
- Set PWRMBASE + offset 31Ch [14,22] = each to 1b For SPT - H
- Set PWRMBASE + 0x31C [15] to 1b if SLP_S0# is enabled
- Set PWRMBASE + 0x31C [13] to 1b if SLP_S0# is enabled and SLP_S0# Voltage Margining is enabled. For SPT – LP
- Set PWRMBASE + offset 320h = 0005DB01h.
- Set PWRMBASE + offset 34Ch [31:30] = 11b
- Set PWRMBASE + offset 34Ch [21] = 1b for SPT-H DMI PHY PG Latency
- Set PWRMBASE + offset 100h [3] = 1b
- Set PWRMBASE + offset 3E8h[22:20] = 010b
- Set PWRMBASE + offset 3E8h[2:0] = 001b
- Set PWRMBASE + offset 600h [31,30] = 11b
- Set PWRMBASE + offset 600h [29] = 1b
- Set PWRMBASE + offset 600h [10:6] = 02h
- Set PWRMBASE + offset 604h [10:6] = 02h
- Set PWRMBASE + offset 3D0h [31,11,10] = 100b
- Set PWRMBASE + offset 3D0h [21:16] = 0Ah
- Set PWRMBASE + offset 3E0h[30] = 1b CPPM Shutdown Qualifier Enable for ModPHY
- Set PWRMBASE + offset 3E0h[29] = 1b ASLT/PLT Selection for ModPHY
- Set PWRMBASE + offset 3E0h[8:0] = 069h LTR Treshold for ModPHY
- Set PWRMBASE + offset 5D0h = 80000001h if not using the PCH debug logic.
- Set PWRMBASE + offset 18h [31, 15, 14, 10, 4] to platform specific.
- Set PWRMBASE + offset 3Ch [23,22] to all 1.
- Set PWRMBASE + offset 3Ch [31:29] for PCH power button override period, 0=4sec, 011b=10sec
- Set PB_DIS bit in PWRMBASE + offset 3Ch [28] = 1b for power button native mode disable designed.

- Set PMC PCI offset ACh [24] = 1b to lock PB_DIS bit to avoid it is changed after boot. Once set, this bit cannot be changed until the next global reset.
- PWRMBASE + offset 3Ch [26] should be set once BIOS is ready to bring DRAM out of reset(handled in SA).
- Set PWRMBASE + offset 48h = 1E2DD701h.
- Set PWRMBASE + offset 4Ch = 00000001h.
- Set PWRMBASE + offset 50h = 1E2DD701h.
- Set PWRMBASE + offset 54h = 00000001h.
- Set PWRMBASE + offset 58h = 0001C000h.
- Set PWRMBASE + offset 68h = 04043400h.
- Set PWRMBASE + offset 80h = BF8FF701h.
- Set PWRMBASE + offset 84h = 0201C7E1h.
- Set PWRMBASE + offset 88h = BF8FF701h.
- Set PWRMBASE + offset 8Ch = 0201C7E1h.
- Set PWRMBASE + offset 98h = 0001C000h.
- Set PWRMBASE + offset A8h = 00181300h.
- Set PWRMBASE + offset ACh = 02006001h.
- Set PWRMBASE + offset B0h = 00181300h.
- Set PWRMBASE + offset B4h = 02006001h.
- Set PWRMBASE + offset C0h = 00060A00h.
- Set PWRMBASE + offset C8h [15] = 1b after this register and register PM_SYNC_MODE (PWRMBASE offset D4h) are programed. PWRMBASE + offset C8h [11:8] are set based on board config.
- Set PWRMBASE + offset D0h = 00200840h.
- Set PWRMBASE + offset D4h = 7000F4E6h.
- Set PWRMBASE + offset E0h = 01040001h.
- Set PWRMBASE + offset E4h = 16BF0003h.
- Set PWRMBASE + offset F4h = 7000F4C0h.
- Set PWRMBASE + offset 200h [31:16] = FFFFh for SPT-LP.
- Once the lanes are assigned as USB3.0, clear corresponding bit in MLSXSWPGP PWRMBASE + offset 200h[15:0] as USB3 lanes do not support SUS Well Power Gating.
- Set MODPHY_PM_CFG2 PWRMBASE + offset 204h [28:24] = 01010b. If DCI is enabled, skip this programming.
- Set PWRMBASE + offset 0E8h [30] = 1b to enable USB2 PHY SUS Well Power Gating.
- Set PWRMBASE + offset 0E8h [8:0] = 18h to set CPU I/O VR Ramp Duration
- Set PWRMBASE + offset 0324h[30] = 1b
- Set PWRMBASE + offset 0340h[30] = 1b
- Set PWRMBASE + offset 03A8h[30] = 1b
- Set PWRMBASE + offset 0324h[8:0] = 006h
- Set PWRMBASE + offset 0340h[8:0] = 008h

- Set PWRMBASE + offset 03A8h[8:0] = 029h

BIOS must set PWRMBASE + 018h [27] = 1b to lock down PMC prior boot to OS for secure purpose.

19.6 Generate SMI on SLP_EN Write

The SMI on Sleep Enable bits are used to generate an SMI#, instead of enter a low power state, when the SLP_EN bit is set, ABASE + 04h[13]. The SMI_ON_SLP_EN is located at ABASE + 30h[4] and the SMI_ON_SLP_EN_STS bit is located at ABASE + 34h[4]. This allows an SMI# to be generated upon setting SLP_EN bit instead of putting the system to a suspend state.

19.7 PCI Express* Bus PME# Signal Based Wakeup

The PCI Express* bus in PCH-based systems resides behind an internal PCI Express* bridge (Dxx:Fn). The System BIOS should provide the _PRW package for PME under that PCI Express* Bus.

```
//    PCH Bus#0, Device #28, Function 0, PCI Express bridge
         Device(RP0n)
         {
                   Name(_ADR, 0x01c000n)

                   Method(_S0D, 0)  {Return (0x00) }
                   Method(_S1D, 0)  {Return (0x00) }
                   Method(_S3D, 0)  {Return (0x00) }
                   Method(_S4D, 0)  {Return (0x03) }

         Name(_PRW, Package(){0x09, 0x3})                    // for PCI PME# event}
         // End of PCH PCI Express bridge
```

19.8 RTC Considerations

19.8.1 RTC Resets

The PCH will set the RTC_PWR_STS bit (PMC PCI offset A4h[2]) when the RTCRST# pin goes low. The System BIOS shouldn't rely on the RTC RAM contents when the RTC_PWR_STS bit is set. BIOS should clear this bit by writing a "0" to this bit position. This bit isn't cleared by any reset function.

19.8.2 UIP Considerations

Reading the RTC requires polling the Update in Progress (UIP) bit in the RTC register 0Ah[7] to ensure the update will not occur for the next 488 us.

Note: UIP SMI has been defeatured since Skylake/Kaby Lake PCH

19.8.3 Alternate Access Mode

Alternate access mode can be used by the System BIOS to read back the data of write only registers, and to restore the value to certain read only registers, which is useful when saving/restoring register content for power management implementations. To enter this mode, set the AME bit,PCR[ITSS] + 31FCh [17].

When in Alternate Access Mode, RTC index registers at I/O port and 72h can be read back by software. I/O port 74h/76h can be read without the AME and thus can be used to read the I/O port 70h [6:0] and 72h [6:0] content.

To read NMI_En bit, since RTC shadows the NMI_EN# bit, then regular IO Reads to Port 70 with Alternate Mode Enabled will be able to read NMI_En as well.

- If AME = 0, RTC Port 70h bit [7:0] returns value 0.
- If AME = 1, RTC Port 70h bits [7:0] returns previously written value. RTC shadows Port 70h bit 7 (NMI_EN#). The regular IO Read to Port 70h always return the shadowed NMI_EN# in RTC.

For more details on using Alternate Access mode, refer to the PCH EDS.

19.8.3.1 Alternate Access Mode Usage for Timer (8254)

If the ALT Access Mode is entered and exited after reading the registers of the PCH timer (8254), the timer starts counting faster (13.5ms). The following steps listed below can cause problems:

1. System BIOS enters ALT Access Mode for reading the PCH timer related registers
2. System BIOS exits ALT Access Mode
3. System BIOS continues through the execution of other needed steps and passes control to the OS.

After getting control in step#3, if the OS does not reprogram the system timer again the timer ticks may be happening faster than expected. For example DOS and its associated software assume that the system timer is running at 54.6ms and as a result the timeouts in the software may be happening faster than expected. Other OS's for example: Microsoft* Windows* family OSes reprograms the system timer and hence do not run into these problems.

So the System BIOS should restore the timer back to 54.6ms before passing control to the OS.

If the System BIOS is entering ALT Access Mode before entering the suspend state it is not necessary to restore the timer contents after the exit from ALT Access Mode.

19.9 Wake Events in the Intel PCH

The following registers contain wake event enable bits to control which events will wake the system when in a low power state:

- PM1_EN, ABASE + 02h
- GPE0_EN, ABASE + 9Ch

Refer to the PCH EDS for more details. The power button is always enabled as a wake event. If the power button is held for more than 4 seconds, the PCH clears the PWRBTN_STS bit, sets the PRBTNOR_STS bit, the system transitions to the S5 state, and only PWRBTN# is enabled as a wake event.

In the ACPI FACP table, ACPI GPE0_BLK should be reported as ABASE + 80h and GPE0_BLK_LEN as 16.

Note: Most wake events will generate an SCI when SCI_EN is enabled or an SMI# if SCI_EN is not enabled, on resume. For some GPI wake events the corresponding bits in the GPIRouteSCI and GPIRouteSMI register fields determines the generation of an SCI or SMI. For wakeup from S5, the System BIOS should clear the status bit corresponding to the wake event before enabling the SCI or SMI to prevent the SCI/SMI# generation. For wakeup from S3-S4 in an ACPI system, System BIOS should not clear the status bit corresponding to the wake event before returning control to the OS.

19.10 Deep S3 / S4 / S5 Power Policies

Deep S3/S4/S5 is an optional feature to reduce power consumption and requires additional hardware design support. The System BIOS can perform the following register programming guidelines to enable system enter Deep S3, Deep S4 or Deep S5 as early as possible during boot.

For platforms that enable Deep S3/S4/S5, BIOS must program SLP_SUS# PWRMBASE + 18h[19:18] stretching to be greater than or equal to the largest stretching value on any other SLP_* pin (SLP_S3#, SLP_S4# or SLP_A#).

GPP_A_13 is reserved for SUSWARN usage for Deep Sx capable platform. If not, GPP_A_13 is default to native mode, and BIOS is required to reconfigure to GPIO if GPIO usage is needed.

Only very few wake events are supported in Deep S4/S5, please check EDS for Deep S4/S5 wake events. If there's any required wake event support on the platform, please check the EDS to determine whether Deep Sx should be enabled or not for that wake event.

Table 19-2: Supported Deep Sx Power Policy Configurations

	DPS3_EN_DC 28h[1]	DPS3_EN_AC 28h[0]	DPS4_EN_DC 2Ch[1]	DPS4_EN_AC 2Ch[0]	DPS5_EN_DC 30h[15]	DPS5_EN_AC 30h[14]
Deep Sx disabled	0	0	0	0	0	0
Enabled in S5 (Battery mode)	0	0	0	0	1	0
Enabled in S5	0	0	0	0	1	1
Enabled in S4 and S5 (Battery Mode)	0	0	1	0	1	0
Enabled in S4 and S5	0	0	1	1	1	1
Enabled in S3, S4 and S5 (Battery Mode)	1	0	1	0	1	0
Enabled in S3, S4 and S5	1	1	1	1	1	1

Note: Mobile platforms support Deep S3/S4/S5 in DC ONLY and Desktop platforms support Deep S3/S4/S5 in AC ONLY.

19.11 Suspend Handler Considerations

19.11.1 S3 (Suspend to RAM) Steps

On a resume from the S3 state, the processor starts executing from the reset vector location of 0FFFF_FFF0h.

The System BIOS executes the following generic steps when resuming a system from the S3 state:

1. Program the base address for ABASE (PMC PCI offset 40h).
2. Verify that the system is resuming from the S3 state by looking at the resume status (WAK_STS, ABASE + 0 [15]) and (SLP_TYP, ABASE + 04h[12:10]). Only execute the steps below, if the system is resuming from the S3 state.
3. Check the power failure bits that follow to make sure that the system can be resumed in a reliable manner: PRBTNOR_STS (ABASE + 00h[11]), PWR_FLR (PMC PCI offset A4h[1]), and PWROK_FLR (PMC PCI offset A2h[0]). If any of these bits are set, then the memory contents cannot be relied upon. So the System BIOS should execute a CF9h reset and go through a normal boot process instead of executing any of the steps described below.
4. The System BIOS will execute the steps described in the Processor SA BIOS Writer's Guide for restoring memory in the platform and continue with the rest of the steps required for a system-wide resume. Refer to Processor SA BIOS Writer's Guide for details.
5. Restore the processor.
   a. Load the processor microcode patch.
   b. Relocate the processor SMM base address from 3000:8000h to the SMM base that is being used in the system.
   c. Initialize the processor L1 and L2 cache. This will involve the restoration of the MTRR registers in the processor.

Note: For multi-processor system, add the following steps:

1) Wakeup the additional processors by using the INIT-SIPI-SIPI sequence as described in the processor BIOS Writer's Guide.
2) Program the MTRR's and processor L1 and L2 cache for additional processors.
3) Load the Microcode patch for additional processors.
4) Program the relocation SMI base address for additional processors.
5) Leave the additional processors executing a HLT

6. Restore the PCI Config registers for the Processor SA devices, PCH devices and the PCI/PCIe* cards present in the system.
7. Restore the Super I/O logical devices.
8. Power up the hard drives and execute any video initialization necessary to get the display memory functional.

9. If the S3 resume is for an ACPI environment, then control is passed to the wake vector at this point. If the S3 resume is for a non-ACPI environment start executing the next step.
10. Restore the 8259 PIC, 8254 Timer, and DMA devices to their original values. These registers must have been saved using Alternate Access Mode before entering the suspend state.
11. Restore the processor SMM save area.
12. Execute the RSM command from the S3 SMM handler.

19.11.2 Restoring Of PCI Registers after a Resume from STR

The Suspend and resume from the S3 state (STR) causes an assertion of the PCIRST# signal and clears the programmed values of all the chipset registers in the core well. As a result, the values must be restored fully on the resume sequence for proper system operation.

It is highly recommended that in a non-ACPI operating system environment that the System BIOS assumes responsibility for ensuring the chipset is fully configured before transferring control to the OS. During the resume sequence from a power managed state like S3 (STR), the System BIOS must fully initialize the chipset and core devices, this includes but is not limited to the PCI configuration header registers (PCI Configuration space 04-3Fh for all PCI devices and functions in the system), internal PIRQ routing registers, edge/level control registers (I/O register 4D0h~4D1h), etc.

Note: This is not an exhaustive list of registers to be restored after a resume from suspend state as each OEM platform may have other devices present in the system.

19.12 Power Optimizer Considerations (MB Only)

Power Optimizer is a new holistic approach that attempts to shift the focus from local power optimizations of one or more components to that of the entire platform. It aims at greatly reducing platform idle power and significantly reducing platform active power. As such it impacts many areas and subsystems (devices, busses, chipset logic, processors, voltage regulators, OS, software applications, etc.) and inherently needs the cooperation of all these participants to be a coherent framework that achieves the maximum power savings benefit.

The System BIOS must perform register programming to enable Power Optimizer in various devices. The registers must be restored during an S3 resume. The registers must be initialized by BIOS for hot plug devices, if OS supported PCI Express Native PM is not enabled. Please refer to Skylake/Kaby Lake Silicon Initialization Code for details information.

19.13 PCI Express Output Clock Enable / Disable Control

To allow maximum power savings in the PCH by shutting down the ICC PLL, as part of runtime D3 entry/exit for PCI Express devices, BIOS must turn off the output clock enable for any PCIe devices that will remain powered, but in reset while in runtime D3. BIOS must clear the enable bit(s) for the output clock enables associated with any device(s) that meet the conditions above (leaving all other clock enable bits at

their previous values) during D3 entry, and restore them to their previous values as part of D3 exit.

Note: It is assumed that BIOS comprehends the PCIe device to output clock mapping. See below for the register location/format used to control the output clock enables:

Note: Write 0000XXXXh to register PCR[ICC] + 100Ch bits [15:00] to force output clocks going to inactive state (See further note below). BIOS must only set the mask bit(s) for the clock(s) associated with devices entering runtime D3 that will keep their CLKREQs asserted (powered but in reset). All other bits must be left at their previous values. As part of exiting runtime D3, before de-asserting the reset to the device, BIOS must clear the mask bit for the clock associated with that device.

Note: It is assumed that the corresponding "Enable Control CLKREQ" bit field for the particular clock source has been set through FITC. Applicable only to the device that supports CLKREQ.

19.14 LPC Power Saving Consideration

To achieve maximum power saving, system BIOS should enable LPC CLKRUN# by set LPC PCI offset E0h[0] = 1b.

However, some old LPC devices might not support it, please contact LPC device vendor to know device capability.

19.15 SPI/eSPI Power Management Programming

BIOS must follow below setting for SPI/eSPI power management configuration

1. Set PCR[ESPISPI] + C004h[29,10:8,5:0] = 1b, 111b,11111b to enable dynamic clock gating.
2. Set PCR[ESPISPI] + C008h[2:0] = 000b to enable power control.

19.16 PSTH Power Management Programming

PSTH is PCH-Specific Transaction Handler(PSTH) which is independent IOSF primary agent and that would be relative with IO strap. BIOS must program properly setting to achieve power saving during POST and restore on S3 resume.

1. In early BIOS phase –
    a. Set PSTHCTL.PSTHIOSFPTCGE PCR[PSTH] + 1D00h[2] = 0b to disable PSTH IOSF Primary Trunk Clock Gating.
    b. Set PSTHCTL.PSTHIOSFSTCGE PCR[PSTH] + 1D00h[1] = 1b to enable PSTH IOSF Sideband Trunk Clock Gating.
    c. Set PSTHCTL. PSTHDCGE PCR[PSTH] + 1D00h[0] = 1b to enable PSTH Dynamic Clock Gating.

2. In late BIOS phase, before booting to OS –

    a. Set PSTHCTL.PSTHIOSFPTCGE PCR[PSTH] + 1D00h[2] = 1b to enable PSTH IOSF Primary Trunk Clock Gating.

§ §

20 TCO Support

20.1 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, ABASE + 30h[13], and GBL_SMI_EN, ABASE + 30h[0], bits are set. The System BIOS must handle this SMI by clearing the TCO_STS bit, ABASE + 34h[13].

If the TCO watchdog timer is not reloaded for another 2.4 seconds and the No Reboot, NR, bit is not set, then PCH will generate a PCI reset.

After booting, the System BIOS must disable the TCO logic within 4.8 seconds or this reset will occur. The System BIOS should disable the TCO watchdog timer from resetting the system by setting the NR bit. The OS or device driver will be responsible for resetting this bit after the TCO driver loads.

The System BIOS must always service the SMI by clearing the TCO_STS and the TIMEOUT bit, TCOBASE + 04h[3], whenever TCO SMIs are enabled. The System BIOS must not reset the watchdog timer in the SMI handler, this must be left to the OS present TCO driver. The System BIOS may log the TCO timeout, if desired.

The TCO_TMR_HALT bit, TCOBASE + 08h[11], if set can halt the TCO timer. 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 System BIOS 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. Please see the Microsoft design specification for details on the format of the WDTRT.

20.1.1 Second Timeout STS

The SECOND_TO_STS bit is set by the PCH to indicate that the TIMEOUT bit has been set or is currently set and a second timeout occurred before the TCO_RLD register, TCOBASE + 0h was written. During POST the System BIOS should make sure that the SECOND_TO_STS bit, TCOBASE + 06h[1], is cleared before transferring control to the OS. The System BIOS should also make sure that the SECOND_TO_STS bit is cleared before initiating any SMBus transfers in the system. If the SECOND_TO_STS bit is set, it prevents the SMBus Block transfers from happening.

20.2 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 System BIOS 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 System BIOS knows the system reboot occurred due to an OS lockup and the third TCO timeout has NOT occurred yet. The System BIOS should clear the SECOND_TO_STS bit and then work to prevent TCO timeouts.

The System BIOS is counting down since the default is TCO timer on. The System BIOS 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 + 08h[11].
2. Set the TCO_TMR, TCOBASE + 12h, to a larger max count (some value larger than the default value, which is 04h or ~2.4 seconds) and then force a TCO timer reload (by writing to the TCO_RLD register, TCOBASE + 00h) regularly.

20.3 Intruder Detect

If required by the OEM, the intruder detect SMI can be used to generate a SMI that would then put the system into a soft off state if the user opens the cover. The INTRUDER# signal sets the INTRD_DET bit, TCOBASE + 06h[0], when detected. The System BIOS should place the system into an S5 state or handle the event appropriately if an INTRUDER# event happens. To enable intruder detect, set the INTRD_SEL field, TCOBASE + 0Ah[2:1] to select SMI and enable the SMI by setting the TCO_EN bit.

20.3.1 TCO Message

After initializing the TCOBASE register, the TCO_MESSAGE1 and TCO_MESSAGE2 (TCOBASE + 0Ch and 0Dh) should be written to indicate boot progress to system manageability code. The values to write are left to the OEM.

20.4 Century Rollover

PCH provides a century rollover status bit, NEWCENTURY_STS (TCOBASE + 04h[7]) to indicate when the year rolls over from 99 to 00. The century rollover status bit can be enabled to generate an SMI if the system is in the G0 state by setting the TCO enable bit in ABASE +30h[13].

After an RTC battery failure the century rollover status bit is in an indeterminate state. To reset the century rollover status bit, the System BIOS must first set and clear the SET bit, RTC register Bh[7] and then clear the NEWCENTURY_STS bit.

The century rollover status bit is not a resume event and must be checked by the System BIOS after booting or resuming from an Sx state. The System BIOS should enable the SMI and update RTC memory when the century rollover SMI occurs.

20.4.1 Additional SMI handling of NEW CENTURY of Skylake/Kaby Lake

There is additional requirement to clear the SMI status of NEW CENTURY in Skylake/Kaby Lake.

1. Detect NEWCENTURY_STS, if set
    a. Clear PCE.HAE, PCR[SMB] + 10h [5] = 0b

b. Loop to write clear NEWCENTURY_STS and check it till it's cleared.
c. Clear TCO_STS.
d. Set PCE.HAE, PCR[SMB] + 10h [5] = 1b

§ §

21 Legacy Free Systems

21.1 Legacy Free Systems Handling by BIOS

"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. On top of this, from Skylake PCH onwards, the port 61h bit 4 would stop toggle and the legacy DMA device is removed in which the I/O ports (81h-83h, 87h, 89h-8Bh, 8Fh) related to it are removed as well. Typically these devices are not truly Plug and Play (PnP) compliant because the BIOS has to explicitly report to the OS about the usage of resources (I/O, Interrupt, DMA etc.) for these devices. This section discusses some of the BIOS implications when dealing with "Legacy Free Systems"

21.2 Virtual Legacy Wire (VLW) Messages

The PCH replace supports for legacy sideband interface signals with VLW messages as the method of conveying the status of the legacy sideband interface signals to the Processor SA. These signals are as follows:

- SMI#, NMI, INIT#, INTR, FERR#
- VLW are inbound messages to the processor. They are communicated via Vendor Defined
- Message over the DMI link. Legacy processor signals can only be delivered via VLW inPCH. Delivery of legacy processor signals (SMI#, NMI, INIT#, INTR, FERR#) via I/O APIC controller is not supported.

It is recommended System BIOS to program the VLW registers as follows:

- P2SB PCI offset 50h = 00F8h (default)

21.3 Reset Mechanism

One of the ways of generating a reset is through the keyboard controller (Super I/O). With the removal of Super I/O the keyboard controller is no longer available. But there are various ways available in PCH for resetting the system.

- I/O Port 92h mechanism (INIT)
- I/O Port CF9h mechanism (INIT and HARD RESET)

The optional ACPI reset mechanism provides a standard mechanism for completely resetting a computer. When implemented, this mechanism must completely reset the entire system—processors, core logic, all buses, and all peripherals. It must be equivalent to power cycling the machine.

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.

ACPI has the requirement that the reset register must only exist in I/O space or in PCI Configuration space on a function in bus 0. Therefore, the Address_Space_ID value in RESET_REG must be set to either I/O space or PCI Configuration space (with a bus number of 0). As the register is only 8 bits, Register_Bit_Width must be 8 and Register_Bit_Offset must be 0.

For PCH-based systems, the RESET_REG should be reported as “CF9h” and the RESET_VALUE should be reported as “06h”.

The value of “0Eh” in I/O Register CF9h toggles the SLP_S3# and SLP_S5# signals, so the System BIOS should use extreme caution based on the hardware board design before reporting a value of “0Eh” as the RESET_VALUE.

Refer to the ACPI Specification for more details.

21.3.1 Fixed ACPI Description Table Boot Architecture and Flags Description

This set of flags is used by an operating system to guide the assumptions it can make in initializing hardware on Intel Architecture (IA-PC) 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-PC 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 the ACPI specification.

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

21.4 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 to the Debug Port Specification for more details.

21.5 Summary of Legacy-Free Handling in ACPI

The System BIOS 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 BIOS 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 System BIOS should use the use the USB legacy Keyboard and Mouse Software IRQ generation support as described above in the USB chapter.
- BIOS 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

21.6 Port 61h Bit 4 Non-toggling Implication

The BIOS, including the build in option rom should not use the port 61h bit 4 toggling as delay routine mechanism. All such usages should be replaced by using other timer.

For system that still need to use add-in card which oprom contains port 61h bit 4 toggling usage, an SMI emulation method is provided to support system that expect this bit to toggle, albeit at a much lower toggling speed, in the range of ~145us (pending on implementation and further improvements) instead of the 15us. It is important to note however, while this emulation can also take care of the internal option rom usage, it will not be able to handle the port 61h bit 4 toggling usage within SMI handler.

Note: If the emulation is on, and some other SMI handler need to perform read to port 61h for non-bit4 stuff, those handler need to turn off the port 61h trap temporarily and re-enable the trap back after read.

Note: Port 61h read has also been used for general I/O read delay without depending on the bit 4 toggling behavior. It is recommended to remove such usage or the delay time resulted from it will increase significantly if the emulation is turned on.

Note: The recommendation is not to turn on this emulation unless necessary. If the impact is limited to the oprom, the emulation must be turned on during the post, and turned off at ExitBootServices assuming no OS usage on that.

21.7 Legacy DMA Removal Implication

The legacy DMA device should not be reported under the ACPI, and all the I/O ports (81h-83h, 87h, 89h-8Bh, 8Fh) related to it must not be used as they are not implemented.

§ §

22 Intel® Serial I/O Bus

Skylake/Kaby Lake adds the Intel® Serial I/O, which consist of controllers as described in table below.

Table 22-1 Intel® Serial IO (SKL/KBL PCH-LP)

Description[7]	_HID	_CID
I2C0 Controller	INT3442	
I2C1 Controller	INT3443	
I2C2 Controller	INT3444	
I2C3 Controller	INT3445	
I2C4 Controller	INT3446	
I2C5 Controller	INT3447	
UART0 Controller	INT3448	
UART1 Controller	INT3449	
UART2 Controller	INT344A	
SPI0 Controller	INT3440	
SPI1 Controller	INT3441	

Note: The DMA is now part of the controllers, except that I2C4 , I2C5 and UART2 do not include the DMA. Please refer to PCH EDS for details.

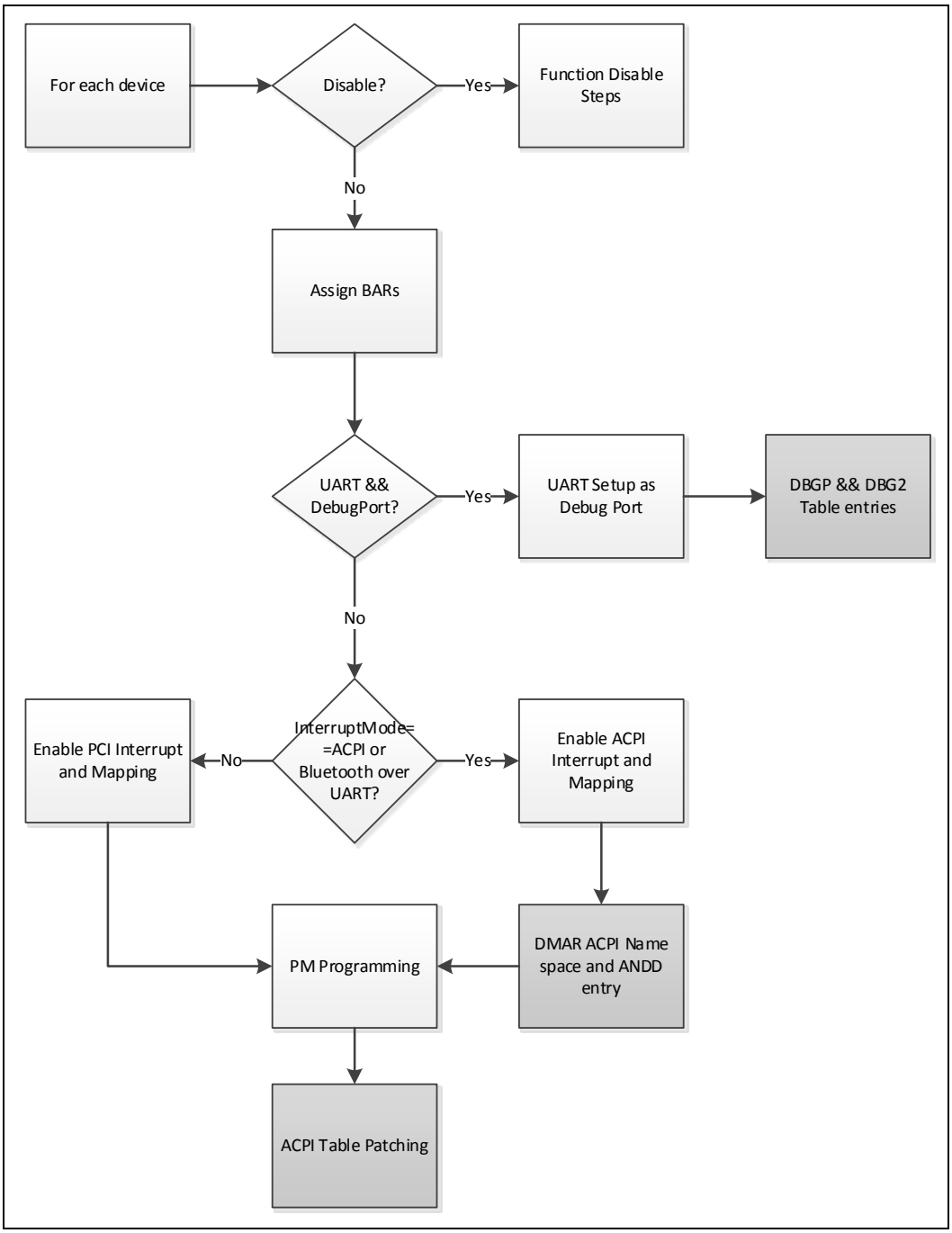
Each controller supports PCI mode and ACPI mode of operation. In PCI mode each controller is host visible via the PCI configuration space. This is the default mode of operation from S5/S4 exit to S0 entry. In ACPI mode the PCI configuration space of the controller is hidden, but the resources are exposed/declared to the OS through ACPI table. There are also some special mode, UART debug mode which expose itself to OS through DBGP/DBG2 table; hidden mode which hide the device completely from the OS, fully managed by the BIOS/ASL code.

Each controller has BAR0 and BAR1 registers. Memory access to the BAR0 register accesses the Controller's MMIO register space. Memory access to the BAR1 register accesses the PCI configuration space of the controller. Note, BAR1 access in PCI mode is not supported. In addition a read should always be performed following a BAR1 access in ACPI mode.

[7] For latest Device:Function definition please refer to SKL PCH-LP volume one EDS.

22.1 Intel® Serial I/O Initialization Flow

Figure 22-1: Intel® Serial I/O Initialization Flow



The BIOS flow for each of the Intel® Serial I/O controller are as follows:

1. The System BIOS may disable the controller prior to PCI enumeration. If disable then below steps should be skipped.

2. The sysem BIOS assigns memory resource to the devices' BAR0 and BAR1. Refer to section 5.9.1.1 on the recommendation.
3. The System BIOS must perform the PM, etc programming.
4. The System BIOS must perform the Interrupt programming. Intel® Serial I/O controllers cannot share IRQs between each other. Refer to section 6.1.2.
5. For UART that is to be used as OS debug port, perform the needed setup.
6. The System BIOS would then perform the late initilization steps that would involve the ACPI tables.

Note: System BIOS should only perform 32-bit accesses to MMIO (except when UART is configured as debug mode) and PCI registers of the Intel Serial I/O controllers.

Note: For ACPI devices that is capable of DMA, system BIOS needs to setup the needed DMAR entries.

22.2 Disabling Intel® Serial I/O Controllers

By default all Intel® Serial I/O controllers are enabled. However, the system BIOS can disable the Intel® Serial I/O Controllers individually per table below. If platform does not utilize Serial I/O controller for connecting peripheral devices it is recommended to disable that controller. (Exception if there are other device locations in the same device number, which the function 0 cannot be disabled. SCS devices need UART0 to be enabled)

1. Prior to setting the D3 bits the BIOS must set, "D0I3MAXDEVPG bits 16 (PMCRE), 17 (IE3), (18 PGE)", Dx:Fx:A0h [18, 17, 16] to [1, 1, 1] in order for Power gating to Function properly.
2. Set the Dx:Fx:84h[1:0] = 11b in D3 HOT State.
3. Read back PME Control Status Dx:Fx:84h
4. Set the Function disable bit of the controllers as indicated in the below table.
5. Set the relevant static PG bit in PWRMBASE + 624h.

Controller	Description
I2C0	PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[I2C0][8] = 1b
I2C1	PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[I2C1][8] = 1b
I2C2	PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[I2C2][8] = 1b
I2C3	PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[I2C3][8] = 1b
I2C4	PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[I2C4][8] = 1b
I2C5	PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[I2C5][8] = 1b
UART0	PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[UART0][8] = 1b
URAT1	PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[UART1][8] = 1b
UART2	PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[UART2][8] = 1b

Controller	Description
SPI0	PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[SPI0][8] = 1b
SPI1	PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[SPI1][8] = 1b

Refer to section 5.3 for additional PSF programming steps.

Note: If Devicexx:Function 0 is disabled then system BIOS must disable rest of Devicexx functions to be compliant with PCI specification. Thus, as long as least one of the functions is to be enabled (in other words , Intel® Serial I/O is not fully disabled/hidden) that PCI Function 0 is always visible.

For Skylake/Kaby Lake PCH-H, similar controllers implemented as in Skylake/Kaby Lake PCH-LP but not connected externally, nor to the PSF primary interface. Those controller function need to be put into D3 using sideband mechanism; after perform a write via sideband mechanisim, BIOS also need to perform a read using sideband mechanisim to avoid ordering and noncoherency problems.

The related controllers that need to be put under D3 are I2C4, and I2C5. BIOS also need to set its PMC static PG bit, PWRMBASE + 624h [5:4] to 11b unconditionally. Refer to section 5.3.1 for related details.

Note: SerialIO does not support PME.

22.2.1 Static Power Gating and Re-enabling of Intel® Serial I/O Devices

Intel® Serial I/O devices can be static PG'ed when all its devices are to be disabled. Refer to section 5.3.1 on its handling.

22.3 Intel® Serial I/O BAR Resource Allocation

The System BIOS must assign address space for BAR0 and BAR1 in each of the controllers. Because of certain OS caching ACPI entries from previous boot, BIOS has to use fixed addresses for these BARs to avoid potential miss match of the BAR addresses and the ACPI entries (when device is under ACPI mode) in certain boot. Fixed address spaces are allocated per section 5.9.1

To keep the flow simple, BIOS will be assigning fixed address to these BARs regardless of the device being ACPI mode or PCI mode. It is alright to use the fixed address during BIOS phase even if the device is PCI mode. If the resource assigning steps are done before BIOS PCI enumeration phase, the same resource assigning steps would be repeated after the BIOS PCI enumeration phase to override the changes done during the BIOS PCI enumeration phase.

22.4 UART OS Debug Port Support

It might be desirable to use the UART as OS debug port for Win7, or even Win8 and other OSes. For Win7 support, we must declare DBGP table to describe the UART controller. It is alright to declare both DBGP and DBG2 table for it to support different OS requirements.

The UART device under this debug mode will go through the same initialization flow as the ACPI mode, but with the 16550 compatible mode turn on (see next paragraph). Besides, the usual Intel Serial UART ACPI device declaration is not needed or simply its _STA to return 0. Furthermore, the related UART memory resource will be claim under system device. No D3 set on the related UART controller and thus there could be power implication as the UART is left at D0. Only enable this when needed.

The UART controllers support 16550 compatible mode. For supporting OS using the UART as debug port, perform the following steps:

1. Set SerialIO BAR0 + 200h[31, 30:16, 15:1] to 1, C35h, 30h
2. Set "GPPRVRW7" by
   - PCR[Serial_IO] + 618h[0] to 1 for UART0, or
   - PCR[Serial_IO] + 618h[1] to 1 for UART1 or
   - PCR[Serial_IO] + 618h[2] to 1 for UART2.
3. BIOS must perform a dummy read BAR1 offset 200h after perform GPPRVRW7. This is required for UART 16550 8-bit Legacy mode to become active.

For 32-bit OS support, it is required to set the UART BAR0 to be under 4GB. Because the register is not legacyIO based, it is required to setup the Debug Port Table to describe this. For Win CS support, it is required to have DBG2 table.

Field	Byte length	Byte offset	Description	
Header				
Signature	4	0	'DBGP'. Signature for the Debug Port Table.	'DBPG'
Length	4	4	Length, in bytes, of the entire Debug Port Table.	52d
Revision	1	8	1	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.	

Field	Byte length	Byte offset	Description	
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.	0
Reserved	3	37	Must be 0.	0
BASE_ADDRESS	12	40	The base address of the Debug Port register set described using the Generic Register Address Structure. (ACPI 2.0)	Space ID : 00 [System Memory] Bit Width : 8 Bit Offset : 00 Reserved : 0 Address : UARTx BAR0

22.5 Intel® Serial I/O SPI Additional Programming Requirement

There could be different SPI device with different active polarity (high or low) on its chip select pin. In order to support this difference, the related PCH GPIO pad cannot be programing to SPI chip select mode until the chip select idle polarity is programmed.

For SPI0,

1. program the cs_state and cs_mode, “SPI0BAR + 224h” bit [1, 0] to [idle_polarity, 1]
1. Program GPP_B_15’s pad mode to 1,

For SPI1,

1. program the cs_state and cs_mode, “SPI1BAR + 224h” bit [1, 0] to [idle_polarity, 1]
2. Program GPP_B_19’s pad mode to 1,

Cross reference to section 16.6.2

22.6 Intel® Serial I/O PCI Mode Initialization

Before PCI enumeration, do the following:

1. Disable BAR1 in PSF by its BAR disable bit
2. Disable BAR1 register in Bridge (64-bit BAR, Bit 19:18)
3. Map interrupts from each of the Intel® Serial I/O controller on to INTA, INTB, INTC, or INTD. Each Intel® Serial I/O device interrupt pin (IPIN) could be mapped into one of the INTx (A/B/C/D) in PCI mode by manipulating “PCICFGCTRLx.Interrupt Pin”, PCR[SCS] + (200h-228h) bit [11:8] to allow

maximum flexibility for interrupt routing. It's INTx can map to the IRQ direct by manipulating the "PCICFGCTRLx.PCI_IRQ", PCR[SCS] + (200h-228h) bit [27:20]. Refer to section 6.1.2 for further details and restriction in assignment

If the device is intended for use during BIOS boot, add the below steps:

4. Assign BAR0 and set Memory space Enable
5. Get Controller out of reset by writting: BAR0 + 204h = 7h
6. On resume from S3, clear Memory Space Enable

It is expected the devices under the Intel ® Serial I/O buses continue to be declared as ACPI devices under the root bus where the controller reside, while the Intel® Serial I/O controller themselves using the normal PCI devices declaration (_ADR).

Steps 1 and 2:

Device	Description
I2C0	Program PCR[PSF_3] + " AGNT_T0_SHDW_PCIEN"[I2C0] [18] = 1 Program "PCICFGCTRL1", PCR[SERIAL_IO + 200h[7] = 1
I2C1	Program PCR[PSF_3] + " AGNT_T0_SHDW_PCIEN"[I2C1] [18] = 1 Program "PCICFGCTRL2", PCR[SERIAL_IO] + 204h[7] = 1
I2C2	Program PCR[PSF_3] + " AGNT_T0_SHDW_PCIEN"[I2C2] [18] = 1 Program "PCICFGCTRL3", PCR[SERIAL_IO] + 208h[7] = 1
I2C3	Program PCR[PSF_3] + " AGNT_T0_SHDW_PCIEN"[I2C3] [18] = 1 Program "PCICFGCTRL4", PCR[SERIAL_IO] + 20Ch[7] = 1
I2C4	Program PCR[PSF_3] + " AGNT_T0_SHDW_PCIEN"[I2C4] [18] = 1 Program "PCICFGCTRL5", PCR[SERIAL_IO] + 210h[7] = 1
I2C5	Program PCR[PSF_3] + " AGNT_T0_SHDW_PCIEN"[I2C5] [18] = 1 Program "PCICFGCTRL6", PCR[SERIAL_IO] + 214h[7] = 1
UART0	Program PCR[PSF_3] + " AGNT_T0_SHDW_PCIEN"[UART0] [18] = 1 Program "PCICFGCTRL7", PCR[SERIAL_IO] + 218h[7] = 1
UART1	Program PCR[PSF_3] + " AGNT_T0_SHDW_PCIEN"[UART1] [18] = 1 Program "PCICFGCTRL8", PCR[SERIAL_IO] + 21Ch[7] = 1
UART2	Program PCR[PSF_3] + " AGNT_T0_SHDW_PCIEN"[UART2] [18] = 1 Program "PCICFGCTRL9", PCR[SERIAL_IO] + 220h[7] = 1
SPI0	Program PCR[PSF_3] + " AGNT_T0_SHDW_PCIEN"[SPI0] [18] = 1 Program "PCICFGCTRL10", PCR[SERIAL_IO] + 224h[7] = 1
SPI1	Program PCR[PSF_3] + " AGNT_T0_SHDW_PCIEN"[SPI1] [18] = 1 Program "PCICFGCTRL11", PCR[SERIAL_IO] + 228h[7] = 1

BIOS needs to update corresponding ACPI entry on interrupt number.

22.6.1.1 Intel® Serial I/O PCI Mode Requirements

If any of the non-zero function are used as PCI mode device, function 0 cannot be disabled and cannot be used as ACPI mode device.

Intel Serial I/O Windows* driver will continue to support ACPI mode in Win8.1 only for I2C4 and I2C5 controllers. For remaining controllers only PCI mode is supported. BIOS would only support manual selection (policy based) for the PCI mode.

The connected device(s) that are connected to the Intel® Serial I/O controllers will still needs its ACPI table(s) declaration.

22.7 Intel® Serial I/O ACPI Mode Initialization

For each Dx:Fx Intel® Serial I/O controller BIOS must enable the ACPI mode as follow:

1. PCI Enumeration -> BAR0 and BAR1, including MSE are programmed (Done in earlier step)
2. The system BIOS must enable the Dx:Fx ACPI mode as per below table.
   a. Map interrupts from each of the Intel® Serial I/O controller to an IRQ. Each Intel® Serial I/O device interrupt pin (IPIN) could be assigned an IRQ direct by manipulating the “PCICFGCTRLx.PCI_IRQ”, PCR[SERIAL_IO] + (200h-228h) bit [19:12]. Refer to section 6.1.2 for further details and restriction in assignment.
   b. Enable ACPI interrupt and hide device’s PCI configuration space in OCP bridge by setting the “PCICFGCTRLx.PCI_IRQ”, PCR[SERIAL_IO] + (200h-228h) bit [1, 0] to [1, 1]
   c. Set the related configuration disable bit in PSF
   d. For each Device that will be used by BIOS during POST, program “RESETS”, BAR0 + 204h[2:0] = 111b

Step 2.b and 2.c shown below.

Device	Description
I2C0	PCR[SERIAL_IO] + PCICFGCTRL1[1, 0] = [1, 1] PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[I2C0][0] = 1b
I2C1	PCR[SERIAL_IO] + PCICFGCTRL2[1, 0] = [1, 1] PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[I2C1][0] = 1b
I2C2	PCR[SERIAL_IO] + PCICFGCTRL3[1, 0] = [1, 1] PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[I2C2][0] = 1b
I2C3	PCR[SERIAL_IO] + PCICFGCTRL4[1, 0] = [1, 1] PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[I2C3][0] = 1b
I2C4	PCR[SERIAL_IO] + PCICFGCTRL5[1, 0] = [1, 1] PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[I2C4][0] = 1b
I2C5	PCR[SERIAL_IO] + PCICFGCTRL6[1, 0] = [1, 1] PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[I2C5][0] = 1b

Device	Description
UART0	PCR[SERIAL_IO] + PCICFGCTRL7[1, 0] = [1, 1] PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[UART0][0] =1b
UART1	PCR[SERIAL_IO] + PCICFGCTRL8[1, 0] = [1, 1] PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[UART1][0] = 1b
UART2	PCR[SERIAL_IO] + PCICFGCTRL9[1, 0] = [1, 1] PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[UART2][0] = 1b
SPI0	PCR[SERIAL_IO] + PCICFGCTRL10[1, 0] = [1, 1] PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[SPI0][0] = 1b
SPI1	PCR[SERIAL_IO] + PCICFGCTRL11[1, 0] = [1, 1] PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[SPI1][0] = 1b

22.7.1 Intel® Serial I/O ACPI Mode Requirements

It is recommended for System BIOS to hide the Intel Serial I/O controllers by returning 00h in the _STA method if Operating System used is not Windows* 8 or above.

In addition:

1. The system BIOS must provide ACPI tables for each of the Intel® Serial I/O controllers as per ACPI 5.0 specification.
2. The system BIOS must provide ACPI table(s) for the connected sensor(s) that are connected to the Intel® Serial I/O controllers.

For ACPI ASL definition please refer to the PCH reference code.

Note: ASL definition must exist even for Serial I/O devices configured in PCI mode. This is because peripheral devices connected to SerialIO (such as touchpads) can only be described in ASL, and they reference their parent controller's ASL definition.

This example describes the ACPI ASL code for an I2C0 Controller with an I0D0 connect sensor.

```
Scope(\_SB.PCI0) {
  Device (I2C0) {
  // for current ACPI ASL definition, refer to PCH reference code
  }
}

//------------------------
// I2C0 connected sensor
//------------------------
Scope(\_SB.PCI0.I2C0) {
    Device(I0D0)
    {
        Name(_HID, "IIC0001")
        Name(_CRS,ResourceTemplate ()
        {
            I2CSerialBus(
                0x15, //SlaveAddress: bus address
                ControllerInitiated, //SlaveMode: default to ControllerInitiated
                100000, //ConnectionSpeed: in Hz
```

```
                AddressingMode7Bit, //Addressing Mode: default to 7 bit
                "\\_SB.PCI0.I2C0", //ResourceSource: I2C bus controller name
            )
        })
    } // Device (I0D0)
} // Device (I2C0)
```

22.7.2 Hidden Mode

If the controller is to be BIOS managed, it would follow all needed ACPI mode programming above, but it would not be reflected as being available to the OS, either without ACPI table for it, or its _STA always return 0.

The memory resource used will be declared under system resource in ACPI.

The controller will be put into D3 prior to OS booting. It is also expected the device be always put into D3 after its usage.

22.7.3 VTd Support

System BIOS needs to provide the DMAR entries for the Intel® Serial I/O devices if they are configured as ACPI mode. Since each of the Intel Serial IO device in Skylake/Kaby Lake PCH is capable of DMA, thus, each of those, when configured as ACPI mode, would have a related device scope and ANDD entry. DMAR entries support for ACPI devices is available from the "Intel® Virtualization Technology for Directed I/O Architecture Specification 2.0".

22.8 Intel® Serial I/O Power Management Programming

22.8.1 Power States

To change the state of each Serial IO device individual _PS0, _PS3 and _PSC ASL methods require implementation. This apllies for both ACPI and PCI mode.

_PS3 method:

1. Set PMECTRLSTATUS Dx:Fx: offset 84h[1, 0] to [1,1] in order to set D3HOT state.
2. Read back PMECTRLSTATUS to avoid ordering and noncoherency problems.

_PS0 method:

1. Set PMECTRLSTATUS Dx:Fx: offset 84h[31:0] to FFFF7FFCh in order to set D0 state. Bits 0 and 1 equal 00b which indicates D0. Bit 15 PMESTATUS is set to 0 as well due to lack of PME support by LPSS.
2. Read back PMECTRLSTATUS to avoid ordering and noncoherency problems.

22.8.2 Clock/PM Initialization

In order for the Clocks, Clock Gating and Power Gating functions to operate properly, the BIOS must set the following CSR bits in the sequence outline below:

3. Set “GPPRVRW2”, PCR[SERIAL_IO] + 604h[11, 1] to [1, 1]
4. Set “PMCTL”, PCR[SERIAL_IO] + 1D0h[5, 4, 3, 2, 1, 0] to [1, 1, 1, 1, 1, 1]
5. For each controller, Set “D0I3MAXDEVPG”, PCI offset A0h[18, 17, 16] to [1, 1, 1]

§ §

23 Intel® Smart Sound Technology (Intel® SST)

The Intel® Smart Sound Technology (Intel® SST) Controller in Skylake/Kaby Lake PCH is a capability block within the HDA device. (Different from the previous PCH where it is a standalone device)

This section describes BIOS requirements for Intel® SST controller support. For anything that is common with the HDA, refer to section 9.

Some of the pins used by this features need to be selected based on the table below:

Pin usage	SKL PCH-LP	SKL PCH-H
HDA	Select through IOBCTL.OSEL	Default
Mobile HDA + I2S1 (SSP1)	IOBCTL.OSEL	N/A
I2S0 and I2S1	IOBCTL.OSEL	N/A
I2S0 (SSP0)	N/A	Select through GPP_D_[5-8]
I2S2 (SSP2)	Select through GPP_F_[0-3]	N/A

23.1 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. Refer to section 9.6.1 for PSF programming needs.

23.2 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[DSP] + F0h [1] being 1.

For SKU that supports Intel® SST, the BIOS can choose to disable it by policy. Refer to section 9.6.2

To enable Intel® SST, the BIOS should:

1. Set the following policies:

   PchPolicy->HdAudioConfig.DspEnable = TRUE;

   PchPolicy->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 Inbox HD Audio driver, BIOS must set the following policy:

PchPolicy->HdAudioConfig.DspUaaCompliance = TRUE;

The policy applies the following programming:

1. Set Sub Class Code to 03h (default), HDA PCI offset 0Ah [7:0] to 03h.
2. Set Programming Interface to 80h, HDA PCI offset 09h [7:0] to 80h, 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.

23.3 Additional Programming Steps

For Skylake/Kaby Lake PCH-LP

If GPP_F_0, GPP_F_1, GPP_F_2 and GPP_F_3 are being used as SSP native function, and controller is not disabled, need to program the related pads' termination to pulldown 20kOhm.

23.4 ACPI Requirements

The offload engine driver invokes the _DSM method to query

1. Non-HD-A Link Descriptor Table
2. Feature Support
3. PP Module Support

Note: The above number is literally the function number used in the _DSM. The UUID for this _DSM is "A69F886E-6CEB-4594-A41F-7B5DCE24C553"

23.4.1 Non-HD-A Link Descriptor Table

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

a. Enable/Disable option in BIOS

b. Link Configurations

c. 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: 01h (Integer)

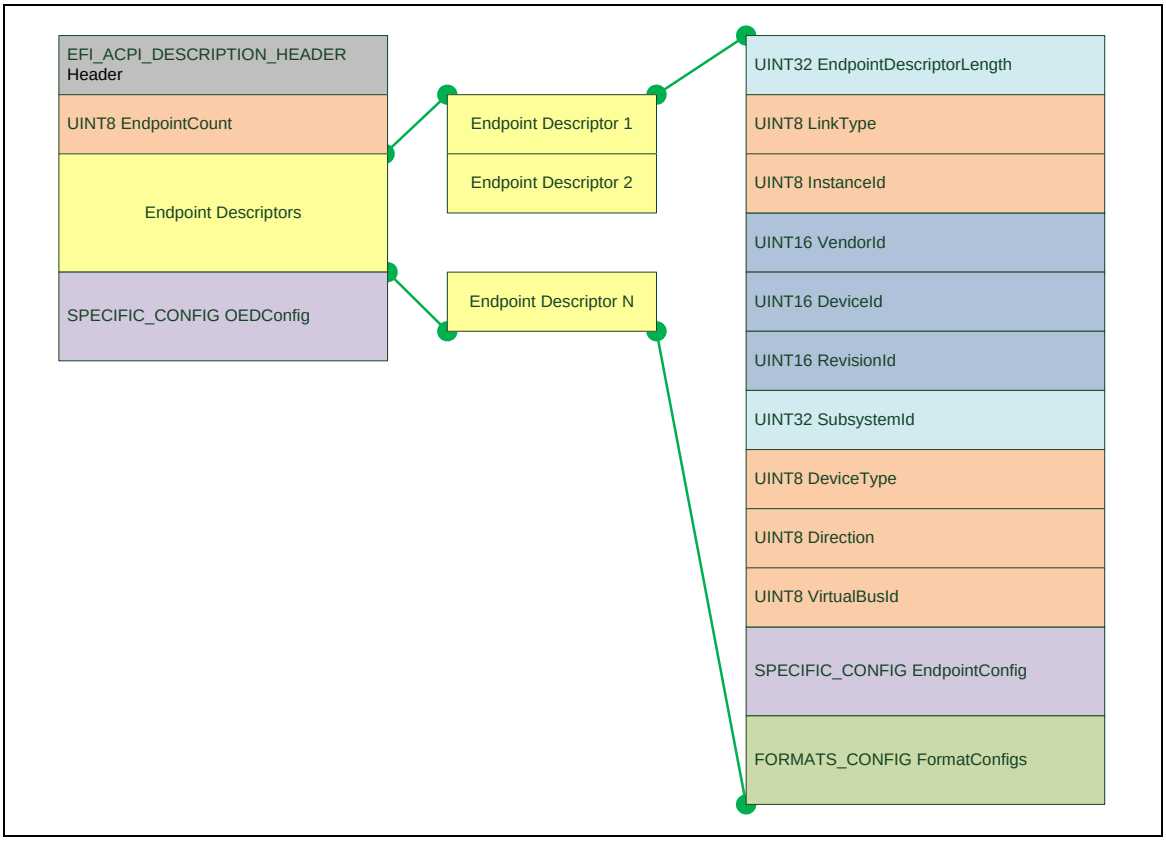
Arg2 — Function Index: 01h (Integer)

Arg3 — Unused

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 23-1: 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.

BIOS enables DMICx2/DMICx4, BT, I2S endpoints (exposes its configuration in NHLT), by PchPolicy, HdaConfig settings:

```
  UINT32  DspEndpointDmic      :  2;    ///< DMIC Select
(PCH_HDAUDIO_DMIC_TYPE enum): 0: Disable; 1: 2ch array; 2: 4ch array
(default)

  UINT32  DspEndpointBluetooth :  1;    ///< Bluetooth enablement: 0: Disable
(default); 1: Enable

  UINT32  DspEndpointI2s       :  1;    ///< I2S enablement: 0: Disable
(default); 1: Enable
```

23.4.2 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	
0	WoV
1	BT sideband
2	codec based VAD
3-4	RSVD
5	BT Intel HFP SCO
6	BT Intel A2DP
7	DSP based speech pre-processing disabled
8	1= Windows Voice Activation. 0= Intel Wake on Voice
9-31	RSVD

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

23.4.3 3rd Party IP

To enable support for 3rd Party IP modules, the following ACPI method has to be added in Platform ACPI code:

```
//
```

```
  // Pre/Post processing modules (3rd Party IP) support for
Intel Smart Sound Technology
  //
  Method(PPMS, 1, Serialized)
  {
    // Helper platform method for HDAS._DSM Function 3 -
Query Pre/Post Processing Module Support
    // Arg0 (equals Arg3 passed to _DSM) - UUID: Specifies
the UUID of the 3rd Party IP module to check (Buffer)
    // Return - TRUE if PP Module supported, else FALSE.

    //
    // Example (to be updated with real GUIDs of supported
3rd party IP):
    // Implement for all supported Pre/Post Processing
modules: 1-N
    //

    // 3rd Party DSP Processing Module #1
    // Check support for 3rd Party IP module with GUID
AABBCCDD-EEFF-1122-3344-556677889900
    If (LEqual(Arg0, ToUUID ("AABBCCDD-EEFF-1122-3344-
556677889900"))){
       Return (1)) // Module #1 supported, return true.
    }

    // 3rd Party DSP Processing Module #N
    // Check support for 3rd Party IP module with GUID
AAAABBBB-CCCC-DDDD-EEEE-FFFF00001111
    If (LEqual(Arg0, ToUUID ("AAAABBBB-CCCC-DDDD-EEEE-
FFFF00001111"))){
       Return (1)) // Module #N supported, return true.
    }
  }
```

PPMS() method must be implemented in Platform Package in HDAS ACPI scope, that is: _SB.PCI0.HDAS.PPMS
'If' conditional statements inside PPMS() method need to be updated with appropriate GUID provided by 3rd Party IP module vendor.

For example, for Intel WoV module ACPI entry with valid GUID looks as below:

```
Scope(\_SB.PCI0.HDAS)
{
  Method(PPMS, 1, Serialized)
  {
    // PP Module Intel WoV
    // Intel WoV GUID: EC774FA9-28D3-424A-90E4-69F984F1EEB7
```

```
    If (LEqual(Arg0, ToUUID ("EC774FA9-28D3-424A-90E4-
69F984F1EEB7"))){
      Return(1) // Intel WoV supported (return true)
    }
  }
}
```

PPMS() is called from HD Audio Subsystem _DSM Function 3 method implemented in ACPI Silicon Package.

23.5 Intel Proprietary Wake Support

aDSP can cause system to wake and since this is an Intel-specific wake source that is not generically supported by Microsoft, aDSP driver will evaluate _DSM below to do the custom handling for Intel wake sources.

The GUID for the Generic _DSM is defined as follows:

{1730E71D-E5DD-4A34-BE57-4D76B6A2FE37}

Refer to the “System BIOS Support for the Intel® HID Event Filter Driver - Developers Guide” for further details.

§ §

24 Intel® Rapid Storage Technology (Intel® RST) for PCIe* Storage

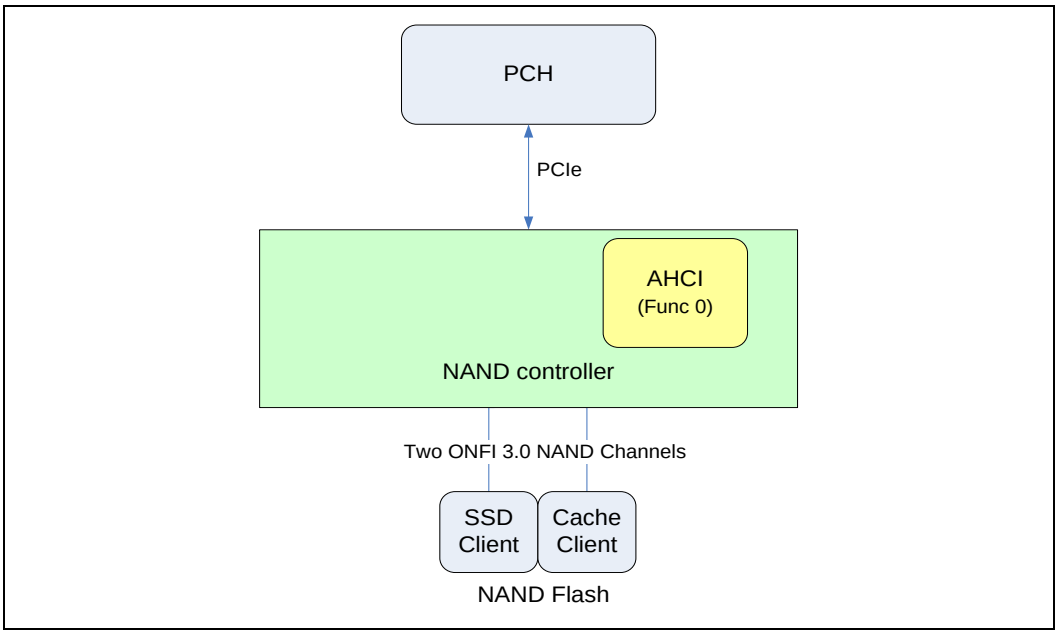
24.1 PCIe* Storage Overview

PCIe* storage refers to supporting storage device on the PCIe interface. It will be enumerated accordingly behind the PCIe root port. The PCIe storage device is exported as an AHCI (device managed) or NVMe controller (standalone) with a single port. Intel RST driver (or a 3rd party driver) may utilize the NVM exported as either an SSD or a cache.

24.2 Remapping and Cycle Router

The PCIe* storage device exports an AHCI controller as function 0 or alternatively an NVMe controller. The AHCI controller has a single port that emulates a Serial ATA-based SSD to ensure that it works with standard AHCI driver software. The PCIe* storage device may be a device managed solution, where NAND flash management is performed in a microcontroller on the PCIe* module. Figure 24-1 describes a device managed solution reporting as an AHCI controller.

Figure 24-1: PCIe* Storage, reporting as AHCI single port device

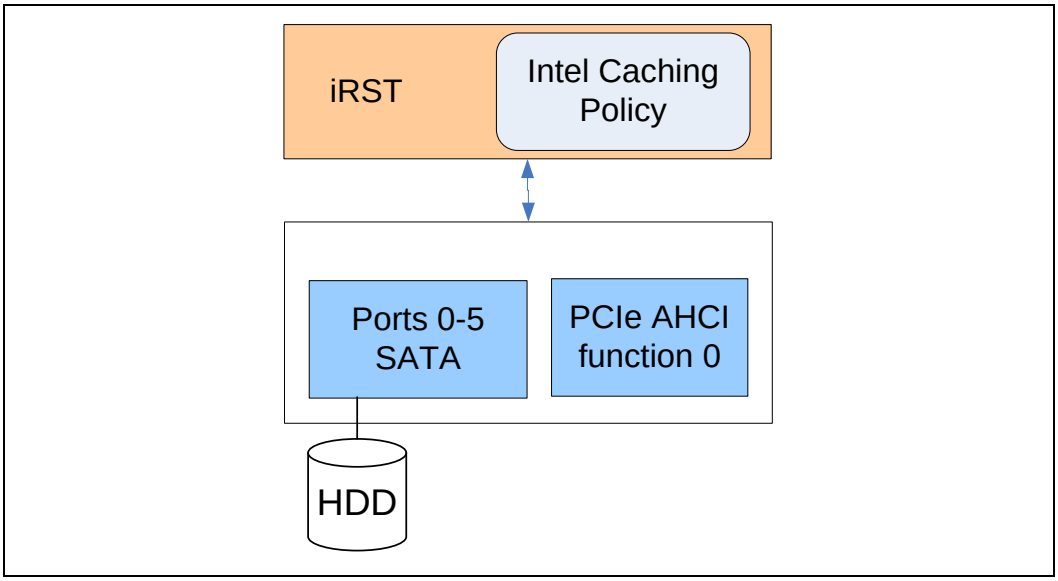


Skylake/Kaby Lake PCH must support remapping the AHCI/NVMe function to be accessible by and controlled by the Intel RST driver instance that is loaded against the PCH's integrated AHCI controller.

When remapping is enabled, the OS sees a combined view of the PCH AHCI integrated controller and the PCIe* AHCI controller (or PCIe* NVMe controller). The remapped view of the AHCI integrated controller is shown in Figure 24-2.

Note: PCIe* root port has been hidden by the PCH.

Figure 24-2: OS View of the Remapped PCIe* Storage



With the root port hidden, the PCIe* AHCI function does not have an OS visible PCI/PCIe* configuration space. All configuration space writes to the PCH integrated AHCI controller must also be issued by the Cycle Router to the PCIe* storage device. This ensures that the configuration of the PCIe* AHCI function mirrors the configuration of the PCH integrated AHCI controller

24.3 Discovery and Initialization

BIOS is responsible to configure the Cycle Router and to enable the register remapping appropriately. Intel® RST for PCIe* Storage device remapping is only supported if the SATA is in RAID mode. Hot plug feature for PCIe* NAND is not supported and must be disabled prior to enabling Intel® RST for PCIe* Storage device remapping feature.

24.3.1 PCH SKU Detection

BIOS needs to detect the PCH SKU to make sure the Intel® RST for PCIe* Storage device remapping feature is supported before executing the initialization steps.

Prior to any PCIe* device enum (our PCIe* init),

1. Set Cycle Router Global Control, SATA PCI offset 0FC0h[1:0] to ‘00b’ to select 1st router.
2. Set the "Port Configuration Check Disable" bit, Sata PCI offset 304h [31]
3. Set “CRE”, SATA PCI offset 300h bit [0] to 1 using byte access

4. Read back the bit 0 value.
5. Write the value into SATA PCI offset 0FC0h[16+n], n=0 for CR1, 1, for CR2 and 2 for CR3.
6. Clear “CRE”, SATA PCI offset 300h bit [0] to 0 using byte access
7. Clear the "Port Configuration Check Disable" bit, Sata PCI offset 304h [31]
8. Repeat step 1-7 with other cycle router; ‘01b’ to select 2nd router or ‘10b’ to select 3rd router.

24.3.2 Initialization

The following describes the initialization flow how BIOS detects and configures the NAND Cycle Router.

1. At the same programming where the “Alternate ID Enable”, and “AIE0 DevID Selection” value are decided, purely based on the BIOS policy in NAND remapping support and PCH SATA mode being RAID:
   a. if supported,
      i. byte write SATA PCI offset 9Ch[7:6, 4:3, 2:0] to [desired value, 00b, 110b]
      ii. If register under ABAR is being referenced after this step, disable MSE, assign 512KB aligned address to ABAR, and enable MSE.
   b. else,
      i. byte write SATA PCI offset 9Ch[7:6, 4:3, 2:0] to [desired value, 10b, 000b]
      ii. Disable RST remap decoding
         (1) Program "NVM Remap Memory BAR Enable", ABAR + A4h[0] to '0b'.
         (2) Program “NRMO”, ABAR + A4h[27:16] = ‘0x00F’, and “MSL” ABAR + A4h[12:1] to all ‘0s’
2. For each NAND Cycle Router, program "Extended General Configuration Register", SATA PCI offset 354h bit [19,17,16] to [1b, 1b, 1b]
3. The BIOS does a mini PCIe* enumeration or manual selection to search for the PCIe* NAND controller that exports an AHCI/NVMe controller function.
   a. Read SATA PCI offset FC0h bit[18:16] to know which CR# are fused enabled.
   b. Refer to the table below to determine PCIe* root port that supports remapping.

Bit	SKL PCH-LP	SKL PCH-H
[4:1]	RP1-RP4	RP1-RP4
[8:5]	RP5-RP8 (CR2)	RP5-RP8
[12:9]	RP9-RP12 (CR3)	RP9-RP12 (CR1)
[16:13]	NA	RP13-RP16 (CR2)
[20:17]	NA	RP17-RP20 (CR3)

Note: Different from earlier PCH, the SATA PCI offset 304h[20:5] indicates lanes that are capable, not ports.

Note: CR# has implication on address offset.

c. Check for mismatching configuration

   i. Set PCIe* Lane Selected, SATA PCI offset 300h[B:A] to 1s, where B:A are the lane that occupied by the particular RP with reference to the above table. For RP7 that is X2, B:A would be 8:7.

   ii. Clear PCCS, SATA PCI offset 304h[30] by writing 1 and then read PCCS. Clear PCIe* Lane Selected.

   iii. If PCCS is one, remove the RP# from the supported port.

d. For each PCIe* root port that supports remapping, query for mass storage class code,01h and

   i. subclass code of AHCI, 06h at offset 0Ah and PI of 01h

   i. subclass code of RAID ,04h at offset 0Ah and VID of 8086h

   ii. subclass code of NVM, 08h at offset 0Ah and PI of NVMe, 02h

e. If none matching, proceed to next root port that supports remapping. If matched, noted down the root port number and proceed to the next root port in next x4 lane.

4. If no PCIe* storage device is detected, or being policy/fuse disabled,

   a. Program “NVM Remap Memory BAR Enable”, ABAR offset A4h [0] to [0b].

   b. Disable Rst remap decoding

      i. Program "NVM Remap Memory BAR Enable", ABAR + A4h[0] to '0b'.

      ii. Program “NRMO”, ABAR + A4h[27:16] = ‘0x00F’, and “MSL” ABAR + A4h[12:1] to all ‘0s’

   c. For each CR, program SATA PCI offset 300h [31:29] to ‘111b’

   d. Exit here.

5. Disable MSE, assign 512KB aligned address to ABAR, and enable MSE.

6. Read “Ports Implemented”, ABAR + offsets 0Ch and keep the value for later step usage.

7. For each port as detected in 3.d step, before BIOS sets the link disable bit of the associated PCIe* root ports, BIOS must discover capability of the PCIe* NAND card supported.

   a. If the PCIe* NAND card is using AHCI register interface, map its BAR5 by setting the “Memory BAR Remap Configuration”, SATA PCI offset 310h[31, 20, 19:16] to [1, 0, 1001b]. if the PCIe* NAND card is using NVMe register interface, map its BAR0 by setting the SATA PCI offset 310h[31, 20, 19:16] to [1, 1, 0100b].

      i. Write all 1s into the PCIe* NAND device MBAR, read it back and keep the value for later steps. Clear the PCIe* NAND device MBAR.

      ii. Check for Pcie NAND Device BAR size and make sure BAR size is <= 16KB

iii. Skip this device from remapping support if the device BAR size > 16KB

b. Read PCIe* NAND Card first capability pointer register at Bx:D0:F0 offset 34h. Read the Capability ID and program to NAND Cycle Router capability remapping register as following.

i. If Capability ID is 11h (MSI-X),

1. Read the PCIe* storage device MSI-X capability DW0[26:16] for table size. Calculate the starting and ending vector needed based on the table size. For first PCIe* storage with MSI-X support, the starting vector is 1, and ending vector is (1 + table size). For the next PCIe* storage with MSI-X support, add earlier PCIe* storage device MSI-X table size to both calculations. If the ending vector value is overflowed, i.e. > 2047, free up earlier allocated resource if any, skip this port from remapping support. Keep the starting vector and ending vector for reference in later step.

2. Program table offset, table BIR to “MSI-X Table Remap Configuration”, SATA PCI offset 340h [31:03, 2:0] accordingly

a. If the table BIR is not the same as the main BAR (BAR5 for AHCI, BAR0 for NVMe),

i. Calculate the BAR size needed. Note: while worst case for the table itself is 32kB, but we cannot guarantee if it is not shared with other vendor defined range or PBA.

ii. Allocate the memory needed from PCH reserved memory range (which also guarantee the address range would not be clashing with the eventual OS/HW assigned BAR5(AHCI)/BAR0(NVMe) memory range, refer to section 5.9.1. If running out of resource, skip this port from remapping.

iii. Program the “MSI-X Table Base Address Register”, SATA PCI offset 344h[31:01] to the allocated value and set its bit[0].

iv. Cross check with step 29

3. Program PBA offset, PBA BIR to “MSI-X PBA Remap Configuration”, SATA PCI offset 348h [31:03, 2:0] accordingly

a. If the PBA BIR is not the same as the main BAR (BAR5 for AHCI, BAR0 for NVMe), and not the same as table BIR,

i. Calculate the BAR size needed. Note: while worst case for the table itself is 256B, but we cannot guarantee if it is not shared with other vendor defined range.

ii. Allocate the memory needed from PCH reserved memory range (which also guarantee the address range would not be clashing with the eventual OS/HW assigned BAR5(AHCI)/BAR0(NVMe) memory range, refer to section 5.9.1. If running out of resource, skip this port from remapping.

iii. Program the “MSI-X PBA Base Address Register”, SATA PCI offset 34Ch[31:01] to the allocated value and set its bit[0].

iv. Cross check with step 30

4. Program the Capability offset to “MSI-X Capability Remap Configuration”, SATA PCI offset 33Ch [23:16] and set bit [31].

c. If Capability ID is 12h (SATA), program the Capability offset to SATA PCI offset 338h [23:16] and set bit [31].If PCIe* NAND card supports Capability ID 12h, set the “I/O BAR Remap Source Configuration” register, SATA PCI offset 320h[31]

d. Read BCC, SCC, PI from the PCIe* NAND and keep the values for later step.

8. Set PCIe* NAND discovered root port respectively to “PCIe* Lane Selected”, SATA PCI offset 300h [20:1]. Each bit represents a lane. (Eg. Bit1 is lane 0). Notes: if x2 port, need ‘11b’ encoding. If x4 port, need ‘1111b’ encoding.

9. Clear GSR “Port Configuration Check Status”, SATA PCI offset 304h[30] by writing 1 to it.

10. Perform steps as indicated by section 24.3.5.

11. Disable link and root port where PCIe* NAND card reside

    a. Set “Link Disable” bit and poll for “Link Active” bit till it is 0.

    b. Set related PSF function disable bit for root port 1, PCR[PSF_1] + “AGNT_T1_SHDW_PCIEN"[RPx][8] to 1

12. Refer to “Ports Implemented” value captured from step 6, program it to the “Shadowed AHCI Ports Implemented”, SATA PCI offset 358h

13. Program “NVM Remapped Memory Offset”, “Memory Space Limit” and “NVM Remap Memory BAR Enable”, ABAR offset A4h [27:16, 12:1, 0] to [10h, 1EFh, 1b]. (MUST only perform this step once)

14. Program SATA PCI offset 300h [0] to ‘1b’ to enable NAND Cycle Router.

15. Read and write back "ASSEL" and "MSS" field , SATA PCI offset 9Ch[4:3,2:0] using byte access

16. Set BME of root port BME bit, SATA PCI offset offset 104 [2] to 1

17. Set all "Memory Base" bits in MBL, PBML and PMBU32 to 1

    a. SATA PCI offset 120h[15:4] to all 1s

    b. SATA PCI offset 124h[15:4] to all 1s

    c. SATA PCI offset 128h [31:0] to all 1s

18. Re-enable PCIe* link by clearing “Link Disable”, SATA PCI offset 150h [4] to ‘0b’.

19. Repeat steps 7 - 18 to configure the next PCIe* NAND device discovered. Need to set Cycle Router Global Control, SATA PCI offset 0FC0h[1:0] to ‘1b’ to select 2nd router or ‘10b’ to select 3rd.

20. Poll for link active for the remapped port(s) and then wait 100ms (tunable). (For optimization, refer to 24.3.3)

21. Replace MSI capability with MSI-X capability in the SATA capability link list

    a. Program the “Next Pointer” in MSI-X identifiers, Sata PCI offset D0h bit [15:8] to 70h (PCI Power Management Capability ID)

22. For each remapped port(s), select it through Cycle Router Global Control, SATA PCI offset 0FC0h[1:0], repeat steps 23 to 30 below (for fast boot handling, refer to 24.3.3)
23. Perform further endpoint programming. Refer to section 24.3.5. Disable MSE, read and write back ABAR, then enable MSE.
24. If endpoint supports MSI-X,
    a. Use the starting and ending vector from step 1. Program them to "MSI-X Starting Vector", ABAR + (808h + n*80h)[10:0] and "MSI-X Ending Vector", ABAR + (808h + n*80h)[26:16] respectively.
    b. Enable Interrupt delivery to MSI-X by programming the "interrupt Delivery", ABAR + offset (808h + n*80h + 4)[31:30] accordingly.
25. Else, program "Device MSI-X Configuration", ABARr + offset (808h + n*80h)[31:0] and ABAR + offset (808h + n*80h + 4)[31:0] to 0.
26. Program "Device Class Code", ABAR + offset (800h + n*80h) with the 32-bits data as below:
    a. Refer to CC, SCC, PI from step 7.d, set up bit[23:0] accordingly
    b. If AHCI is detected, also set Bit 31.
27. Read root port's "Negotiated Link Width" and "Current Link Speed", SATA PCI offset 152h bit [9:4] and [3:0], and store these link info in 9-bits field:
    a. BIT[2:0] for Cycle Router 1, bit[0] for link speed and bit[2:1] for link width
    b. BIT[5:3] for Cycle Router 2, bit[3] for link speed and bit[5:4] for link width
    c. BIT[8:6] for Cycle Router 3, bit[6] for link speed and bit[8:7] for link width

The bit definition as below:

i. Link Speed (1 bit): 0b = GEN1/GEN2, 1b = GEN3

ii. Link Width (2 bits): 0b = x1, 1b = x2, 2b = x3

28. Refer to endpoint MBAR writable bits in step 7.a.i, program the value into the "Device Memory BAR Length", ABAR + offset (804h + n*80h).
29. If the device supports unique MSI-X Table BAR, program the related PCIe* storage device BAR with the same value as step ii, and 0 to the higher memory BAR if it is 64-bit BAR. Needs to handle this during the RTD3 flow as well.
30. If the device supports unique MSI-X PBA BAR, program the related PCIe* storage device BAR with the same value as step ii, and 0 to the higher memory BAR if it is 64-bit BAR. Needs to handle this during the RTD3 flow as well.
31. For each enabled CR(s), program SATA PCI offset 300h [31:30] to '11b'. Then, for each disabled CR(s), program SATA PCI offset 300h[31:29] to '111b'. (They must be performed separately, eg: perform the programming required for all enabled CR(s) first, then only perform the remaining programming for all disabled CR(s))
32. Program the enabled CR information into "Remap Capability Register", ABAR + 800h[2:0]If any vendor specific register above 40h is programmed during

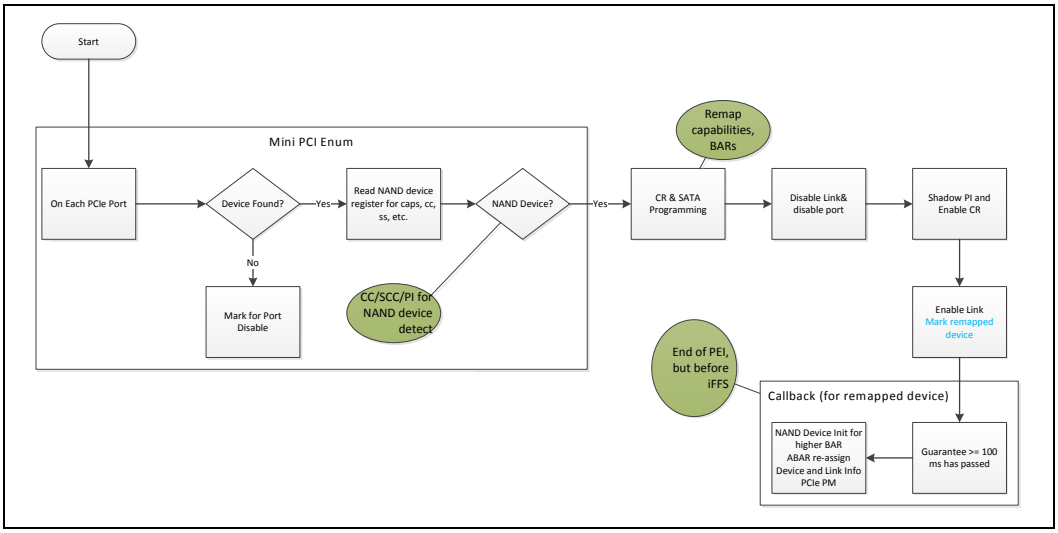
post, they would need to be restored in the RTD3 ASL code. Nothing is from this from the reference code end.

33. Program “AHCI MSI-X Configuration”, ABAR + 808h[8:0] with remapped device link info value captured in step 27.
34. For each CR(s), program Extended General Configuration Register SATA PCI offset 354h[20]=0b to allow legacy interrupts delivery.

24.3.3 Fast Boot/Sx Resume Handling

For fast S3/S4/S5, steps 20-33 to be split for late execution. At end of step19, current timer value is to be recorded. A later callback (such as End of PEI event) will check the recorded timer and guarantee at least 100ms has passed before proceed performing steps 20-33 . The 100ms is tunable parameter, which it could be increased to cover the time needed for the polling for link active, can also be reduced to tune for a given device that is capable of responding to configuration access sooner. It is important to know that AHCI Bar -based registers are not accessible before the callback which would re-assign address to ABAR.

Figure 24-3: Fast Boot/Sx Resume Handling



24.3.4 Manual Port Selection

The above steps help automatically locate the PCIe* storage device for remapping. However, if there are more PCIe* storage device connected than what we can remap or just for the sake of skipping enumeration on known configuration, user might wish to choose which PCIe* storage device to remap and which to be used as standalone. The user selection however, must comply with the rule in which there could only be one remapped PCIe* storage device per x4 lane.

24.3.5 PCIe* Capabilities Enabling

Since the root port would be function disable, it would be the BIOS role to enable the PCIe link power management features. These include ASPM control and L1 substate support in both root port and endpoint.

Before perform the link disable or power off (RTD3) step,

1. If endpoint supports L1 Sub-States,
   a. Save endpoint's L1 Sub-States Extended Capability offset + 0Ch[7:0] value
   b. Save endpoint's L1 Sub-States Extended Capability offset + 08h[31:0] value
2. Save endpoint's Link Control's "Enable Clock Power Management" , "Common Clock Configuration" and "Active State Link PM Control" fields, PCIe* Capability offset + 10h[8, 6, 1:0]
3. Save endpoint's Max_Payload_Size field, PCIe* Capability offset + 8h[7:5]
4. Save LTR context
   a. Root port's LTR Enable, RP PCI offset + 68h[10]
   b. If endpoint support LTR,
      i. Endpoint's LTR Enable, endpoint PCI Express Capabilities offset + 28h[10]
      ii. Endpoint's LTR Capability offset + 4h[31:0]

After the link re-enabled and the 100ms (tunable) delay, programming done through the CAIR and CADR pair

1. Clear all BARs within endpoint to 0.
2. Restore endpoint's L1 Sub-States Extended Capability offset + 0Ch[7:0]
3. Restore endpoint's L1 Sub-States Extended Capability offset + 08h[31:29, 25:16]
4. Restore endpoint's L1 Sub-States Extended Capability offset + 08h[3:0]
5. Restore LTR context
   a. Root port's LTR Enable, RP PCI offset + 68h[10]
   b. Endpoint's LTR Enable, endpoint PCI Express Capabilities offset + 28h [10]
   c. Endpoint's LTR Capability offset + 4h[31:0]
6. Restore endpoint's Max_Payload_Size field, PCIe* Capability offset + 8h[7:5]
7. Restore endpoint's Link Control's "Enable Clock Power Management" and "Common Clock Configuration" fields, PCIe* Capability offset + 10h[8, 6].
   a. If CCC is set to 1, then
      i. Perform the link retrain by setting the Retrain Link (RL) bit, SATA PCI offset 150h[5] to 1,
      ii. Poll for link active
8. Restore endpoint's "Active State Link PM Control", PCIe* Capability offset + 10h[1:0].

Note: There is no programming path to non-zero function in the NAND device after remapping. Thus, if the NAND device is a multiple function device, BIOS would not be able to setup common clock, ASPM correctly.

24.3.6 ACPI Requirement

Under the remapped device scope

```
Method (_PS3, 0) {
    //put the endpoint device into D3 using CAIR/CADR pair
}

Method (_PS0, 0) {
    //put the endpoint device into D0 using CAIR/CADR pair
}
```

24.3.7 SATA Controller Dependency

The remapping is using PCH SATA controller interface. Thus this controller has to be enabled for the remapping to work. There are cases that lead to SATA controller to be disabled; either a BIOS option or its “Ports Implemented” register (PI) field is being 0. The PI register field being 0could be caused by no HSIO lane allocated to the SATA controller. When remapping is enabled and device found, the SATA controller should not be function disabled even if the PI==0.

24.3.8 RTD3 Consideration

As seen in initialization steps, there could be configuration registers within the PCIe* storage device need to be initialized. The same requirements are applied during RTD3 exit.

The power gating/ungating of the PCIe* storage device has to be under the SATA adaptor scope because the link must be up while the SATA adaptor is still in D0 and possible SATA register access triggering remap access cycles.

Since we have more than 1 CRs, in the RTD3 restore code we would need to select it thru “Cycle Router Accessibility Select”, SATA PCI offset FC0h[1:0]. The access to the PCIe* storage configuration is also through an index data pair mechanism. To avoid concurrent access, mutex needs to be applied by using a single serialized ASL method for CR access.

24.3.9 SMM Consideration

If there is ever a need to use SMI to access the remapped endpoints or root port configuration space, there CR index and CAIR must be saved and restored

24.3.10 Limitation

The remapping is only supported by the UEFI iRST driver. If legacy Intel RST option rom is used, remapping should not be enabled.

Only UEFI driver are foreseen on the NVMe preboot support. Thus, only UEFI boot could be supported.

24.4 Platform Handling

Must always load UEFI RST Orom if remap is enabled, this also means avoid the native AHCI or AHCI OROM support since the Intel RST will also support the SATA AHCI mode.

24.5 RST One Click Optane Enable

24.5.1 Purpose

The purpose of RST One Click Optane Enable module is to provide interface for OS level applications that allows to automatically configure BIOS to meet Intel Optane requirements.

24.5.2 Mode of operation

To allow for BIOS-OS communication, BIOS creates a UEFI variable that allows OS level application to request 2 operations. First one is to change Integrated AHCI controller mode of operation to RST mode. The second one allows remapping to be enabled on remap capable root ports (i.e. the ones with connected PCIe storage device and with cycle router).

Only one operation can be requested at a time and requesting integrated AHCI controller mode change operation takes precedence over enabling remapping. If OS application requests from BIOS both AHCI controller mode changing and enabling remapping, only AHCI controller mode changing is performed.

After performing each operation BIOS clears the variable (sets all bits to 0).

Presence of RstOptaneConfig variable indicates that BIOS supports RST One Click Enable functionality and should be ready to service OS requests.

24.5.3 Variable definition

Definitions required to create RstOptaneConfig variable.

VariableName: "RstOptaneConfig"

Guid:

{0x4DA4F952, 0x2516, 0x4D06, {0x89, 0x75, 0x65, 0x03, 0x64, 0x03, 0xA8, 0xC7}
}

Structure definition:

```
typedef struct {
  UINT8 RemapEnable : 1;
  UINT8 ModeSwitch : 1;
  UINT8 Reserved : 6;
} RST_CONFIG_VARIABLE;
```

To request mode change of integrated AHCI controller ModeSwitch bit should be set to 1. To request enabling remapping RemapEnable bit should be set to 1. Reserved bits should be set to 0.

24.5.4 Platform prerequisites

Before creating RstConfigVariable BIOS should check whether platform supports RST mode of operation. This is done by calling GetSupportedRstMode () function. This is the only prerequisite that platform must meet. Even if platform has no cycle routers operational it can still create RstConfigVariable as BIOS still supports automatic AHCI controller mode switching.

24.5.5 Synchronization with BIOS setup

To ensure that user has correct settings displayed in BIOS setup at all times a synchronization mechanism has to be provided. Reference implementation modifies setup variables directly so that BIOS setup always reflects actual settings.

24.5.6 Security considerations

Since not all OSes can boot from remapped drives this module makes platform vulnerable to DOS attack in which malware in OS enables remapping on bootable PCIe drive(on Windows this is only possible after malware gets admin privileges). To ensure that recovery from such a scenario is possible it is recommended that BIOS with RST One Click Optane Enable module allows for manual remapping disabling.

§ §

25 Intel® Sensor Hub (ISH)

25.1 ISH Disable/Enable Handling

ISH could be fused disabled, refer to section 5.3.1 for its handling.

If unfused, BIOS can choose to disable it by

1. Set PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[ISH][8] to 1
2. Set PWRMBASE + 620h [5] to 1.

Reset need changing from static PG'ed to enable. Refer to section 5.3.1 for further description.

25.2 PCI/ACPI Modes

While the ISH can be configured to work in either PCI or ACPI modes, the POR is to support PCI mode only. The configuration is done via a write to the IOSF2OCP private register by a sideband access.

25.2.1 PCI Mode of Operation

System BIOS can decide to leave ISH to operate in PCI mode so that operating system can discover the device when doing PCI scan. BIOS have to disable BAR1 by setting PCICFGCTRL.BAR1_DISABLE to 1b in ISH private config space before PCI enumeration.

1. PCICFGCTRL.BAR1_DISABLE, offset (200h) [7] = 1b.
2. Set PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[ISH][19, 18] to [1, 1]

25.2.2 PCI Mode Interrupt Pin Mapping

ISH device interrupt pin (IPIN) could be mapped into one of the INTx (A/B/C/D) in PCI mode by manipulating "PCICFGCTRL.Interrupt Pin", PCR[ISHBR] + (200h) bit [11:8] to allow maximum flexibility for interrupt routing. It's INTx can map to the IRQ direct by manipulating the "PCICFGCTRL.PCI_IRQ", PCR[ISHBR] + (200h) bit [27:20]. Refer to section 6.1.2 for further details and restriction in assignment.

No ACPI entry needed to describe whatever devices connected at the back of ISH. ISH will be able to read HID descriptor from the devices connected to it.

25.3 Power Management Programming

In order for the Clocks, Clock Gating and Power Gating functions to operate properly, the BIOS must set the following CSR bits in the sequence outline below:

1. Set "PMCTL", PCR[ISHBR] + 1D0h[5, 4, 3, 2, 1, 0] to [1, 1, 1, 1, 1, 1]

25.4 Initialization Requirement

After memory is available, set BME bit.

25.5 Intel Proprietary Wake Support

Intel® Sensor Hub can cause system to wake and since this is an Intel-specific wake source that is not generically supported by Microsoft, Sensor Hub driver will evaluate _DSM below to do the custom handling for Intel wake sources.

The GUID for the Generic _DSM is defined as follows:

{1730E71D-E5DD-4A34-BE57-4D76B6A2FE37}

Refer to the "System BIOS Support for the Intel® HID Event Filter Driver (CDI Document# 557003)" for further details.

25.6 MEI MKHI Messages

These MKHI messages are used for communicating ISH requests and responses prior DID handshake.

The feature uses ISH⇔BIOS communication mechanism to unlock PDT update functionality.

Command	Prior DID (BIOS)	After EOP (OS)
_PDT_UNLOCK_DATA	Yes	No

25.6.1 PDT Unlock Data Message

The BIOS based unlock interface is used to implement PDT lock state protect from change by malicious SW. It is assumed that PDT unlock command issued by BIOS provides certain protection level as it cannot be done by malicious SW.

Field	Size (Byte)	Default	Description
MEI Header	4	80160003h	Refer to Skylake ME11 BIOS specification for more details.
ISH Service Message Header	4	00010012h	Refer to Skylake ME11 BIOS specification for more details.
File Name	12		ASCII destination file name (up to not null terminated 12 bytes)
Number of Elements	2	01h	1 element
Data Type	2	01h	01h – PDT_UNLOCK
Payload Size	2	00h	0 – no data

This message requires no response.

§ §

26 Camera I/O Controller (LP)

The CSI-2 HCI brings imaging data from an external imager into the system.

26.1 CIO2 Controller Enabling

1. If CIO2 is initially function disabled (PWRMBASE + 620h [6] = 0b), set PWRMBASE + 620h [6] to 1b and issue reset.
2. If CIO2 is initially function enabled (PWRMBASE + 620h [6] = 1b),
3. For PCH B1 and above, program CIO2 FLIS register PCR[CAM_FLS] + Ah to 00h using sideband access.
4. Set PCR[CAM_FLS] + 13h[26] = 1b to save power using sideband access.

26.2 CIO2 Controller Disabling

1. Put the device into D3 Hot by programming its PMCSR register in CIO2 PCI offset D4h[1,0] = 11b.
2. Set PCR[CAM_FLS] + 13h[26] = 1b to save power using sideband access.
3. Set PCR[PSF_1] + 31Ch[8] = 1b to perform PSF function disable.
4. Set PWRMBASE + 620h [6] = 1b to perform Skycam PMC function disable.

26.3 Device Mode

CIO2 can operate as ACPI or PCIe device (starting SKL PCH-LP C0). When working in ACPI mode the PCI configuration space of the device is hidden, but the resources are exposed/declared to the OS through ACPI table (i. e.interrupt number, BAR address).

26.4 Additional Initialization Programming Steps

1. If CIO2 device has been initially disabled, program PWRMBASE + 31Ch [12] to 1b and move to last setp.
2. If CIO2 device has been initially enabled, program PWRMBASE + 31Ch [12] to 0b.
3. When working in PCIe mode, program CIO2 PCI offset 91h[7:0] to D0h.
4. Program CIO2_BAR. In case when working in ACPI mode refer to section 5.9.1 for the allocation.
5. Enable BME and MSE (CIO2 PCI offset 4h[2,1] = [1,1]).
6. Program CIO2_PCE, CIO2_BAR + 1430h [2,1] to [1,1].
7. Program CIO2_GPR0, CIO2_BAR + 1434h [31: 0] to 01134640h.
8. Program CIO2_GPR1, CIO2_BAR + 1438h [31: 0] to 000031C0h.
9. Move CIO2 to D0i3 state by programming CIO2_D0I3C register. Set CIO2_BAR + 1408h = 4h.
10. Program CIO2_BAR + 1400h[13:10,8,6:1] = 1111b, 1b, 111111b to enable CIO2 clock gating.

11. When working in ACPI mode, hide CIO2 device. Set PCR [PSF_1] + "AGNT_T0_SHDW_CFG_DIS_CIO2" bit [0] to 1.
12. When working in PCIe mode, disable MSE, BME and clear CIO2_BAR.
13. Last step. Lock all lockable fields in CFG space. Set "FNCFG", PCR[CAM_CHC] + Ch [0] to 1b using 8 bit access.

These FLIS registers (IOSF-SB port AAh) have to be written with meaningful values that are platform type specific.

The following is a list of these FLIS registers:

1. CSI_CLKTRIM
1. CSI_DATATRIM
2. CSI_DATATRIM1
3. CSI_CTLE
4. CSI_RCOMP_CONTROL

26.5 ACPI Requirements

SkyCam platform supports one or more (up to 4) imaging sensors.

§ §

27 eSPI

eSPI would be replacing LPC interface, provides LPC compatible host access method through its channel 0. It occupies the same BDF as the LPC (only one can be active at one time, refer to section 2.5.2 for fulfilling this requirement), and thus the PCI configuration access is also the same

On top of the configuration register, eSPI has additional private configuration register for soft straps and slave configuration access. The eSPI Controller maps the Slave's Configuration registers to its IOSF-SB Private Control Register space via a set of keyhole registers. This provides access for debug purposes (via Hotham) as well as to BIOS to allow it to read/modify the eSPI Slave configurations.

27.1 eSPI Enabling

eSPI is enabled using eSPI strap. No further BIOS action on PSF needed, as it is fully handled by HW.

27.2 eSPI Device Interrupt Support

The ACPI method is used to communicate the IRQ number used by the eSPI endpoints. BIOS must assign IRQ to the eSPI endpoint per each endpoint device's IRQ programming requirement. The device can map any IRQ up to IRQ119 (IRQ1,3-7,9-14-119) . If the device has a way to use IRQ 24 to 119 however, the IRQ assignment must be unique comparing to the rest including GPIO, Intel® Serial I/O, SCS, and etc. devices. Refer to section 0 for further details.

27.3 Tunneled Slave Access Programming

The EC configuration registers could be accessed using the Tunneled Slave access mechanism in the eSPI Controller. The 2 eSPI controller registers involved are SLV_CFG_REG_CTL (PCR[ESPISPI] + 4000h) and SLV_CFG_REG_DATA (PCR[ESPISPI] + 4004h).

27.3.1 Get_Configuration

1. Clear status from previous transaction by writing 111b to status in SCRS, PCR[ESPISPI] + 4000h [30:28]
2. Program SLV_CFG_REG_CTL with the right value (Bit[31]=01, Bit [20:19]=<SlvID>, Bit [17:16] = 00b, Bit[11:0] = <addr_xxx>).
3. Poll the SCRE (PCR[ESPISPI] +4000h [31]) to be set back to 0
4. Check the transaction status in SCRS (bits [30:28])
5. Read SLV_CFG_REG_DATA.

27.3.2 Set_Configuration

1. Clear status from previous transaction by writing 111b to status in SCRS, PCR[ESPISPI] + 4000h [30:28]
2. Program SLV_CFG_REG_DATA with the write value.
3. Program SLV_CFG_REG_CTL with the right value (Bit[31]=01, Bit [20:19]=<SlvID>, Bit [17:16] = 01b, Bit[11:0] = <addr_xxx>).
4. Poll the SCRE (PCR[ESPISPI] +4000h [31]) to be set back to 0
5. Check the transaction status in SCRS (bits [30:28])

Note: If a write is to a supported register in the reserved register address range (0h –7FFh), the eSPI-MC updates its local copy of the Slave configuration registers after the write has been successfully sent to the Slave. If the SBLCL bit, PCR[ESPISPI] + 4000h [27] is set however, any write to these range would be blocked.

Note: eSPI-MC does no checking of the register values (even for supported Slave Capabilities / Configuration Registers) – the SW assumes full responsibility for programming legal values supported by both the eSPI-MC and the Slave.

27.3.3 Get_Status

It is similar to Get_Configuration, but only the lower 2 bytes of data are valid and SLV_CFG_REG_CTL bit [17:16] = 10b.

1. Clear status from previous transaction by writing 111b to status in SCRS, PCR[ESPISPI] + 4000h [30:28]
2. Program SLV_CFG_REG_CTL with the right value (Bit[31]=01, Bit [20:19]=<SlvID>, Bit [17:16] = 10b, Bit[11:0] = <addr_xxx>).
3. Poll the SCRE (PCR[ESPISPI] +4000h [31]) to be set back to 0
4. Check the transaction status in SCRS (bits [30:28])
5. Read SLV_CFG_REG_DATA [15:0].

27.3.4 In-band Reset

It is similar to Set_Configuration, but there is no write data and SLV_CFG_REG_CTL bit [17:16] = 11b.

1. Clear status from previous transaction by writing 111b to status in SCRS, PCR[ESPISPI] + 4000h [30:28]
2. Program SLV_CFG_REG_CTL with the right value (Bit[31]=01, Bit [20:19]=<SlvID>, Bit [17:16] = 11b).
3. Poll the SCRE (PCR[ESPISPI] +4000h [31]) to be set back to 0
4. Check the transaction status in SCRS (bits [30:28])

27.4 Initialization Requirement

For the security, the BIOS must set the SBLCL bit, PCR[ESPISPI] + 4000h [27] to 1 after the initial eSPI link configuration.

§ §

28 Storage and Communication Subsystem

The Storage and Communication Subsystem (SCS) consists of eMMC 5.0, SDIO 3.0 and SD Card 3.0 IP.

From a PCI standpoint, eMMC 5.0, SDIO 3.0 and SD Card 3.0 appear to be 3 PCI functions but only 2 can be active at any given time, either eMMC / SDIO or eMMC / SDXc Card. It is up to the BIOS to perform a function disable on the inactive PCI function in the IOSF2OCP Bridge.

The following lists the Device number and Function number for each IP.

Device: Function	Controller	Description
D30:F4	eMMC 5.0 Controller	Used for Storage device
D30:F5	SDIO 3.0 Controller	Used for Legacy Wi-Fi Interface
D30:F6	SD 3.0 Controller (SDXc)	Used for Storage device

28.1 SCS Initialization Flow

The BIOS initialization flow for each of the eMMC/SDIO/SDXc controller is as follow:

1. The System BIOS may disable the controller prior to PCI enumeration. If disable then below steps should be ignored.
2. The System BIOS must perform the Snoop programming.
3. The System BIOS must perform the DLL programming.
4. The system BIOS must perform Power Management and Clock Gating features programming
5. The System BIOS must enumerate the controller in PCI Mode.

28.2 SCS Devices Enable/Disable Handling

If any of the eMMC/SDIO/SDXc devices is enabled, the Intel® Serial I/O UART0 must be enabled as PCI mode.

28.2.1 D30F7 Disable

The D30F7 device is not to be enabled, thus, perform the related PSF programming to disable it.

1. Set PCR[PSF_2] + “AGNT_T0_SHDW_PCIEN”[D30_F7] [8] = 1

28.2.2 SDIO / SD Card

SDIO is de-featured from Skylake platforom, it must be disabled.

1. To enable SD card (or disable SDIO)

a. Set “IOSF2OCP.GPPRVRW2[2]”, PCR[SCS] + 604h bit[2] = 1
b. Program SDXc PCI offset 84h bit[1:0] = 11b (put SDIO into D3)
c. Set PCR[PSF_2] + “AGNT_T0_SHDW_PCIEN”[SDIO] [8] = 1 (disable primary access to SDIO)

2. To disable both SDIO and SD card
   a. Program SDXc PCI offset 84h bit[1:0] = 11b (put SDIO into D3)
   b. Set PCR[PSF_2] + “AGNT_T0_SHDW_PCIEN”[SDIO] [8] = 1 (disable primary access to SDIO)
   c. Program SDIO PCI offset 84h bit[1:0] = 11b (put SD into D3)
   d. Set PCR[PSF_2] + “AGNT_T0_SHDW_PCIEN”[SD] [8] = 1 (disable primary access to SD)

28.2.3 SD Card Specific Programming

BIOS must set Rcomp delay 1p8mode to 10ms by below setting.

1. Set "IOSF2OCP.GPPRVRW6[1P8_SEL_DELAY]", PCR[SCS] + 614h bit[7:0] = 7Fh.

28.2.4 eMMC

To disable eMMC,

a. Set “IOSF2OCP.GPPRVRW2[1]”, PCR[SCS] + 604h bit [1]=1b
b. Set PCR[PSF_2] + “AGNT_T0_SHDW_PCIEN”[eMMC] [8] = 1

28.3 SNOOP / Non-SNOOP Programming

All upstream SCS transaction is snooped. There is no support to issue non snooped upstream access. BIOS must perform snoop programming on Intel® SCS by setting “IOSF2OCP. IOSFCTL [7]”, PCR [SCS] + 00h bit [7] = 1b.

28.4 Power Management and Clock Gating Programming

Set the following registers to configure power management and clock gating:

1. “IOSF2OCP.PMCTL”, PCR[SCS] + 1D0h [5:0] = 111111b

2. “IOSF2OCP.GPRRVRW1”, PCR[SCS] + 600h = 800DE7E7h

3. “IOSF2OCP.GPRRVRW1”, PCR[SCS] + 600h [BIT3] = 1b for PCH-LP stepping C1 onwards

28.5 SD DLL Programming

BIOS must set SD DLL registers to the following values:

1. Set Tx CMD Delay Control 1 Register: SD BAR0 + 820h = 0Ch
2. Set Tx CMD Delay Control 2 Register: SD BAR0 + 80Ch = 22282824h
3. Set Tx Data Delay Control 1 Register: SD BAR0 + 824h = 0A13h
4. Set Tx Data Delay Control 2 Register: SD BAR0 + 828h = 28272824h
5. Set Rx CMD + Data Delay Control 1 Register: SD BAR0 + 82Ch = 07070D0Eh

28.6 eMMC DLL Programming

Every boot, BIOS will read the HS400 DLL policy to determine if DLL programming is required. The two registers to be programmed are:

- eMMC BAR0 offset 824h: Tx Data Delay (HS400), bit[14:8] Tx Data Delay Control 1
- eMMC BAR0 offset 830h: Rx Strobe Delay DLL 1 (HS400 Mode) , bit[14:8] Rx Strobe Delay Control

Tx Data Delay Path register is used to delay the Data relative to the clock. Rx Strobe Delay path register is used to delay the Rx Strobe in relation to the incoming CMD and Data.

BIOS must set eMMC DLL registers to the following values:

6. Set Tx CMD Delay Control 1 Register: eMMC BAR0 + 820h = 0Fh
7. Set Tx CMD Delay Control 2 Register: eMMC BAR0 + 80Ch = 1C282828h
8. Set Tx Data Delay Control 1 Register: eMMC BAR0 + 824h = 090Fh
9. Set Tx Data Delay Control 2 Register: eMMC BAR0 + 828h = 1C272828h
10. Set Rx CMD + Data Delay Control 1 Register: eMMC BAR0 + 82Ch = 1C070B13h
11. Set Rx CMD + Data Delay Control 2 Register: eMMC BAR0 + 834h = 181Ch
12. Set Rx Strobe Delay Control Register: eMMC BAR0 + 830h = A0Ah
13. Set Rx CMD + Data Delay Control 2 Register: eMMC BAR0 + 834h[17:16] = 01b (Rx Clock before Output Buffer)

28.6.1 eMMC HS400 Software Training Sequence

In Skylake PCH-LP, eMMC 5.0 is supported, which supports HS400 mode. If the eMMC device also supports HS400 mode, BIOS needs to perform the training required to achieve this HS400 speed.

The following flow describes the Tx and Rx one-time training needed for running at HS400. At the first boot the BIOS need to read the capabilities of the eMMC device if it is not supporting HS400 the following should be skipped. Every boot the BIOS will read the training policy, if training is not required, BIOS will skip training procedure.

BIOS will also read HS400 DLL policy, if DLL is valid, BIOS will move the eMMC device to operate in HS400 modes.

28.6.2 High Level Flow of eMMC HS400 Training

1. Check training policy:-
   a. If training is not required but DLL is valid, move eMMC device to HS400 mode. Skip the following.
   b. If training is required, continue the following flow.
2. Check if the 512Bytes block write location is not protected.
3. Perform Rx Data Path training.
4. Perform Tx Data Path training.

28.6.3 HS400 Mode Selection

1. Preboot eMMC driver to initialize eMMC device to HS200 mode.
2. Perform HS400 mode selection.
   a. Set HS_TIMING to 1h High Speed interface timing (CMD6). This is required prior to setting bus width to DDR 8 bit.
   b. Set BUS_WITDH to 6h for DDR 8 bit (CMD6).
   c. Check that No Error is reported (CMD13).
3. HS400 mode selection done.

28.6.4 Rx Data Path Training

1. Read a single block data of 512Bytes which is used in HS200 tuning.
2. Move eMMC device to HS400 mode.
3. Set eMMC BAR0 offset 830h: Rx Strobe Delay DLL 1 (HS400 Mode), bit[14:8] to the Minimal Expected Passing Tap (Smin).
4. Set eMMC BAR0 offset 830h: Rx Strobe Delay DLL 2 (HS400 Mode), bit[6:0] to middle point, 13h.
5. Read single block of data.
6. If CRC fails on first read, increase the DLL step and repeat block read until passed, else if CRC passed on first read, decrease the DLL step and repeat block read until failed.
7. Store the Minimal Rx Data Path passing step number (Smin).
8. Set eMMC BAR0 offset 830h: Rx Strobe Delay DLL 1 (HS400 Mode), bit[14:8] to the Maximal Expected Passing Tap (Smax).
9. Read single block of data
10. If CRC fails on first read, decrease the DLL step and repeat block read until passed, else if CRC passed on first read, increase the DLL step and repeat block read until failed.
11. Store the Maximal Rx Data Path passing step number (Smax)
12. Compute the Rx DLL Optimal Point (Sopt) = (Smax – Smin) / 2 + Smin
13. Store the Rx DLL Optimal Point Value (Sopt)
14. Set eMMC BAR0 offset 830h: Rx Strobe Delay DLL 1 (HS400 Mode) , bit[14:8] to the Rx DLL Optimal Point (Sopt)
15. If any of procedure above failed:-

a. Move eMMC device back to operate at HS200 mode.
b. Disable HS400 support by setting eMMC BAR0 offset 814h: HS400 Support (eMMC only), bit[29] = 0b

16. Proceed to Tx Data Path Training

28.6.5 Tx Data Path Training

Once completed and passed the HS400 Rx Data path training, the BIOS will perform the HS400 Tx path training. This is intended to calibrate the Tx DLL. During this training, the BIOS will set the DLL to certain value, perform single block (512Bytes) write to the eMMC, read the CRC status on completion for pass/fail indication. The detail flow is as follows:

1. Read a single block data of 512Bytes which is used in Rx Data Path training.
2. Set eMMC BAR0 offset 824h: Tx Data Delay (HS400), bit[14:8] to the Minimal Expected Passing Tap (Smin)
3. Write single block of data
4. If CRC fails on first write, increase the DLL step and repeat block write until passed, else if CRC passed on first write, decrease the DLL step and repeat block write until failed.
5. Store the Minimal Tx Data Path passing step number (Smin)
6. Set eMMC BAR0 offset 824h: Tx Data Delay (HS400), bit[14:8] to the Maximal Expected Passing Tap (Smax)
7. Write single block of data
8. If CRC fails on first write, decrease the DLL step and repeat block write until passed, else if CRC passed on first write, increase the DLL step and repeat block write until failed.
9. Store the Maximal Tx Data Path passing step number (Smax)
10. Compute the Tx DLL Optimal Point (Sopt) = (Smax – Smin) / 2 + Smin
11. Store the Tx DLL Optimal Point Value (Sopt)
12. Set eMMC BAR0 offset 824h: Tx Data Delay (HS400), bit[14:8] to Tx DLL Optimal Point Value (Sopt)
13. If any of the procedures above failed:-
    a. Move eMMC device back to operate at HS200 mode.
    b. Disable HS400 support by setting eMMC BAR0 offset 814h: HS400 Support (eMMC only), bit[29] = 0b

From this point for every boot the BIOS should check the HS400 Data Valid policy and HS400 Tuning Required policy bit.

If HS400 Tuning Required policy bit is “1” run the above algorithm. Else, if

HS400 Data Valid policy bit is “1” read the stored DLL values from non-volatile variable and write them to Rx DLL and Tx DLL accordingly and move eMMC device to HS400 mode.

Note: The eMMC partition and address for the block reading/writing step is from platform policy. Platform is responsible to save the HS400 Data Valid, HS400 Rx DLL Data and HS400 Tx DLL Data as non-volatile variable to be used for subsequent boots.

28.6.6 Fast Path and S3 Path Consideration

It is expected that the training is only needed to run once for a new BIOS. Subsequently, the trained data (pass or fail) would be reapplied for the rest of boots. Thus, it is expected that the trained data (eMMC BAR0 offset 830h: Rx Strobe Delay DLL 1 (HS400 Mode), bit[14:8] and eMMC BAR0 offset 824h: Tx Data Delay (HS400), bit[14:8]) from section 28.4.4 and 28.4.5 to be saved after first boot for the subsequent reapplying purpose.

28.7 PCI/ACPI Modes

While the SCS devices can be configured to work in either PCI or ACPI modes, the POR is to support PCI mode only, except for the SDIO need to be in ACPI mode when its connected device need SW LTR support. The configuration is done via a write to the IOSF2OCP private register by a sideband access.

28.7.1 PCI Mode of Operation

System BIOS can decide to leave SCS to operate in PCI mode so that operating system can discover the device when doing PCI scan. BIOS have to disable BAR1 by setting PCICFGCTRLx.BAR1_DISABLEx to 1b in SCS private config space before PCI enumeration.

For eMMC

1. PCICFGCTRL1.BAR1_DISABLE1, offset 200h [7] = 1b.
2. Set PCR[PSF_2] + "AGNT_T0_SHDW_PCIEN"[eMMC][19, 18] to [1, 1]

For SDIO

1. PCICFGCTRL2.BAR1_DISABLE2, offset 204h [7] = 1b.
2. Set PCR[PSF_2] + "AGNT_T0_SHDW_PCIEN"[SDIO][19, 18] to [1, 1]

For SDCARD

1. PCICFGCTRL3.BAR1_DISABLE3, offset 208h [7] = 1b.
2. Set PCR[PSF_2] + "AGNT_T0_SHDW_PCIEN"[SDCARD][19, 18] to [1, 1]

28.7.2 PCI Mode Interrupt Pin Mapping

Each eMMC/SDIO/SDXc device interrupt pin (IPIN) could be mapped into one of the INTx (A/B/C/D) in PCI mode by manipulating "PCICFGCTRLx.Interrupt Pin", PCR[SCS] + (200h-20Ch) bit [11:8] to allow maximum flexibility for interrupt routing. It's INTx can map to the IRQ direct by manipulating the "PCICFGCTRLx.PCI_IRQ", PCR[SCS] + (200h-20Ch) bit [27:20]. Refer to section 6.1.2 for further details and restriction in assignment.

28.7.3 ACPI Mode of Operation

Only SDIO is expected to operate in ACPI mode when Wi-Fi device is connected to the SDIO. To operate in ACPI mode, BIOS have to allocate memory resource to the device's BAR0 and BAR1 and followed by setting its MSE bit. Lastly enable ACPI interrupt and hide the device PCI configuration space.

For SDIO

1. Allocate resource to device PCI offset 10h and PCI offset 18h. Refer to section 5.9.1 for the recommendation.
2. Set MSE bit, PCI offset 4h[1]
3. Set PCR[SCS] + 204h[1, 0] = [1, 1]
4. Set PCR[PSF_2] + "AGNT_T0_SHDW_CFG_DIS"[SDIO][0] = 1b

28.7.4 ACPI Mode Interrupt Assignment

Device assigns its IRQ directly by manipulating the “PCICFGCTRLx.ACPI_IRQ”, PCR[SCS] + (200h-20Ch) bit [19:12]. Refer to section 6.1.2 for further details and restriction in assignment.

28.8 SCS ACPI Requirements

BIOS must define each eMMC/SDIO/SDXc connected device as an ACPI device under each related controller scope. Inside that device definition, BIOS must specify a Remove method (_RMV) object that returns 0 to indication that the device cannot be removed for Intel® eMMC device.

In the ACPI device description associated with the Wi-Fi on SDIO device, the BIOS will include an address range of 32-bit memory-mapped I/O (MMIO) descriptors which correspond to the LTR registers in the Skylake PCH LP SDIO Host controller.

The BIOS will also initialize the SW_LTR_VALUE and AUTO_LTR_VALUE registers of the SDIO host controller with the value NOREQ during power-on reset initialization.

28.8.1 Example of ASL code for eMMC/SDIO/SDXc Connected Device

```
Scope(\_SB.PCI0) {
  //
  // SCS eMMC (PCI Mode)
  //
  Device(PEMC) {
    Name(_ADR, 0x001E0004)


    Device (CARD)
    {
      Name (_ADR, 0x00000008)
      Method(_RMV, 0x0, NotSerialized)
      {
        Return (0)
      } // End _RMV
    }
  } // Device(PEMC)
```

```
  //
  // SCS Sdio (PCI Mode)
  //
  Device(PSDO) {
    Name(_ADR, 0x001E0005)
  }

  //
  // SCS Sd Card (PCI Mode)
  //
  Device(PSDC) {
    Name(_ADR, 0x001E0006)
  }

  //
  // SCS Sdio (ACPI Mode)
  //
  Device(SDHB) {
    Name (_HID, "INT344D")
    Name (_CID, "PNP0D40")      // SDA Standard Compliant SD Host Controller
    Name (_DDN, "Intel(R) SDIO Controller - 80869D2C")
    Name (_UID, 1)

    Name (RBUF, ResourceTemplate ()
    {
      Memory32Fixed (ReadWrite, 0x00000000, 0x00000800, BAR0)
      Interrupt(ResourceConsumer, Level, ActiveLow, Shared, , , SIRQ) {22} // SDIO
IRQ
    })
    Method (_CRS, 0x0, NotSerialized)
    {
      CreateDWordField(^RBUF,^BAR0._BAS,BVAL)
      CreateDWordField(^RBUF,^SIRQ._INT,IRQN)
      Store (SDB0, BVAL)
      Store (SDIR, IRQN)
      Return (RBUF)
    }
    Method (_STA, 0x0, NotSerialized)
    {
      If (LOr(LEqual(SDB0, 0), LNotEqual(SDME, 2)))
```

```
      {
        Return (0x0)
      }
      Return (0xF)
    }
  }

  Device(SCSR) {
    Name(_HID,EISAID("PNP0C02"))
    Name(_STA,0x3) // device present and decodes its resources, but not to be
displayed in OSPM
    Name(_UID,6)
    Name (RBUF, ResourceTemplate ()
    {
      Memory32Fixed (ReadWrite, 0x00000000, 0x0000000C, SBA0)  // System Device
Range 0
                                                               // PCH Platform
that is having SDIO WIFI to manipulate SBA0,
                                                               // reducing it's
size to 0x4 if WIFI is connected
      Memory32Fixed (ReadWrite, 0x00000000, 0x000007F4, SBA1)  // System Device
Range 1
      Memory32Fixed (ReadWrite, 0x00000000, 0x00000000, BAR1)  // SCS SDIO BAR1
    })
    Method (_CRS, 0x0, NotSerialized)
    {
      CreateDWordField(^RBUF,^SBA0._BAS,SBV0)
      CreateDWordField(^RBUF,^SBA1._BAS,SBV1)
      CreateDWordField(^RBUF,^BAR1._BAS,BVAL)
      CreateDWordField(^RBUF,^BAR1._LEN,BLEN)

      Store (Add(SDB0,0x800), SBV0)
      Store (Add(SDB0,0x80C), SBV1)

      If (LOr(LNotEqual(SDB1, 0), LEqual(SDME, 2)))
      {
        Store (SDB1, BVAL)
        Store (0x1000, BLEN)
      }
      Return (RBUF)
    }
  }
}
```

§ §

29 HSIO and FIA

29.1 FIA Lane Ownership and Handling

Skylake PCH supports HSIO lanes that can be dynamically mapped to different controllers. Refer to Skylake EDS for more details on HSIO Muxing on LP, S and H SKUs.

It is well possible that a controller/function has no associated lane and thus it needs to be functioned disable for either power reason or specification reason (SATA).

For Skylake PCH-LP

1. Function disable unused root port
   a. Disable PCIe Port 1 if lane 4 not owned by PCIe, PCR[FIA] + 250h bit [19:16] != 0
   b. Disable PCIe Port 2 if PCR[FIA] + 250h bit [23:20] != 0
   c. Disable PCIe Port 3 if PCR[FIA] + 250h bit [27:24] != 0
   d. Disable PCIe Port 4 if PCR[FIA] + 250h bit [31:28] != 0
   e. Disable PCIe Port 5 if PCR[FIA] + 254h bit [03:00] != 0
   f. Disable PCIe Port 6 if PCR[FIA] + 254h bit [07:04] != 0
   g. Disable PCIe Port 7 if PCR[FIA] + 254h bit [11:08] != 0
   h. Disable PCIe Port 8 if PCR[FIA] + 254h bit [15:12] != 0
   i. Disable PCIe Port 9 if PCR[FIA] + 254h bit [19:16] != 0
   j. Disable PCIe Port 10 if PCR[FIA] + 254h bit [23:20] != 0
   k. Disable PCIe Port 11 if PCR[FIA] + 254h bit [27:24] != 0
   l. Disable PCIe Port 12 if PCR[FIA] + 254h bit [31:28] != 0
   m. Perform the function disable handling as stated in section 8.2.1.1
2. Function disable unused GbE
   a. Disable GbE if all of the following conditions are met
      i. PCR[FIA] + 250h bit [27:24] != 11b
      ii. PCR[FIA] + 250h bit [31:28] != 11b
      iii. PCR[FIA] + 254h bit [03:00] != 11b
      iv. PCR[FIA] + 254h bit [19:16] != 11b
      v. PCR[FIA] + 254h bit [23:20] != 11b
   b. Perform the function disable handling as stated in section 10.2
3. Function disable unused SATA
   a. Disable SATA if all of the following conditions are met
      i. PCR[FIA] + 254h bit [11:08] != 10b
      ii. PCR[FIA] + 254h bit [15:12] != 10b
      iii. PCR[FIA] + 254h bit [27:24] != 10b
      iv. PCR[FIA] + 254h bit [31:28] != 10b

v. Intel® RST for PCIe* Storage device remapping is not enabled (Note: might need delay decision handling)

b. Perform the function disable handling as stated in section 14.3

29.1.1 CLKREQ to Root Port Mapping

There are only six PCIe CLKREQ# pins available in PCH-LP but there are 12 possible PCI Express Ports that could be mapped to these 6 CLKREQ# pins. Similarly, there are only 16 PCIe CLKREQ# pins available in PCH-H but there are 20 possible PCI Express Ports that could be mapped to these 16 CLKREQ# pins. The GbE when enabled will use one of the CLKREQ#.

The mapping of the PCIe*Express Ports and GbE to the CLKREQ# pins can be specified through the DRCRM{x} registers.

All RP that does not have CLKREQ# mapping does not need any mapping configuration change, but the particular RP(s)/GbE would be function disabled. Ideally, the BIOS policies that enable/disable RP/GbE should have matched the CLKREQ assignment.

After the configuration done, set PCR[FIA] + 0h bit [31, 17, 16, 15] to [1, 1, 1, 1].

29.1.2 Runtime Trigger

RTD3 component of each controller/function would

1. On entering D3, sets its corresponding bit in MSPDRTReq, PWRMBASE +_208h bit[19:0] and polls for corresponding bit in MSPDRTRAck PWRMBASE +_20Ch bit[19:0] to be set
2. On entering D0, clears its corresponding bit in MSPDRTReq and polls for corresponding bit in MSPDRTRAck to be clear.

For feature such as PCIe* NAND remapping, its RTD3 methods would take care of the controller/functions that it controls.

29.1.2.1 Special PCIe* Hot Plug Case

Under the case where PCIe* hot plug/unplug detection can be done by using external pin such as GPI GPE, BIOS can make use of the particular GPI GPE event to perform the runtime trigger.

1. On detecting slot not populated, set the corresponding PCIe* function's MSPDRTReq bit to '1' and polls for corresponding bit in MSPDRTRAck to be set.
2. On detecting slot populated, clear the corresponding PCIe* function's MSPDRTReq bit to '0' and polls for corresponding bit in MSPDRTRAck to be clear

29.2 MEI MKHI Messages

These MKHI messages are used for communicating HSIO requests and responses prior DID handshake.

Table 29-1: MKHI MEI Messages Availability

Command	Prior DID (BIOS)	After EOP (OS)
HSIO_CMD_REQ	Yes	No
HSIO_CMD_RESP	Yes	No

29.2.1.1 Intel ME Kernel Hos Interface (MKHI) MEI Header

All MKHI related MEI requests and responses define the Intel® MEI messages for host and ME Kernel communication. These messages are used to communicate prior to memory initialization. After the DID message is received, all further communication must be done through the normal HECI messaging interfaces. The data structure for that header is defined below. For more information on this messages please refer to Skylake Intel ME 11 BIOS Specification.

Table 29-2: Definition of MKHI MEI Header

Field	Length (bit)	Value	Description
Group Id	8	F2h	Kernel Client specific information for dispatching
Command	7	N/A	Command code specific to MEI client
IsResponse	1	N/A	Set to 1 in response message
Reserved	8	00h	Reserved
Result	8	N/A	Result of the request message

29.2.1.2 HSIO Command Request (HSIO_CMD_REQ)

If acknowledgement is not received, then the Intel ME is likely in a failure state preventing it from functioning.

Table 29-3: Definitions for HSIO Command Request Message

Field	Length (byte)	Value	Description
MEI Header	4	80040008h	Refer to Skylake ME11 BIOs specification for more details.
MKHI Header	4	000001F2h	Command code specific to MEI client HSIO_CMD_REQ = 01h

29.2.1.3 HSIO Command Response (HSIO_CMD_RESP)

This command is used by Intel ME to acknowledge the BIOS message refer to Table 29-5 for details.

Table 29-4: Definitions for HSIO Command Response Message

Field	Length (byte)	Value	Description
MEI Header	4	80088008h	Refer to Skylake ME11 BIOs specification for more details.
MKHI Header	4	000081F2h	Command code specific to MEI client HSIO_CMD_RES = 81h
HSIO Command Response Data	4	N/A	ME used this message to acknowledges the BIOS message. Please refer to Table 29-5 for definitions on HSIO Command Response Data

Table 29-5: Definition of HSIO Command Response Data

Field	Length (byte)	Value	Description	
CRC	4	Varies	CRC calculated across the HSIO entries.	
Version	2	Varies	Version, chipset, and stepping this table targets	
Family	2	Varies	Bits	Description
			7..4	Product Family 0-LPTH. 1-LPTLP, 2-WPTLP, 3-WBG, 4:7-Reserved, 8-SkylakeLP, 9-SkylakeH, 10-LBG, 11:15-Reserved
			3..0	Stepping

29.3 Power Management Programming

BIOS must enable HSIO Power Gating for all data and common lanes for SPT-LP only.

1. Enable HSIO Power Gating for all data lanes by set
   - HSIO lane#0: PCR[EAh] + 024h[29] =1b
   - HSIO lane#1: PCR[EAh] + 224h[29] =1b
   - HSIO lane#2: PCR[EAh] + 424h[29] =1b
   - HSIO lane#3: PCR[EAh] + 624h[29] =1b
   - HSIO lane#4: PCR[EAh] + 824h[29] =1b
   - HSIO lane#5: PCR[EAh] + A24h[29] =1b
   - HSIO lane#6: PCR[E9h] + 024h[29] =1b
   - HSIO lane#7: PCR[E9h] + 224h[29] =1b
   - HSIO lane#8: PCR[E9h] + 424h[29] =1b
   - HSIO lane#9: PCR[E9h] + 624h[29] =1b

- HSIO lane#10: PCR[A9h] + 024h[29] =1b
- HSIO lane#11: PCR[A9h] + 224h[29] =1b
- HSIO lane#12: PCR[A9h] + 424h[29] =1b
- HSIO lane#13: PCR[A9h] + 624h[29] =1b
- HSIO lane#14: PCR[A9h] + 824h[29] =1b
- HSIO lane#15: PCR[A9h] + A24h[29] =1b

2. Enable HSIO Power Gating for all common lanes
   - PCR[MODPHY0] + 8030h[0] =1b
   - PCR[MODPHY1] + 8030h[0] =1b
   - PCR[MODPHY2] + 8030h[0] =1b
   - PCR[MODPHY3] + 8030h[0] =1b

§ §

30 Intel® Trace Hub (Intel® TH)

Intel ® Trace Hub (Intel ® TH) is a debug architecture that unifies hardware and software system visibility, by improving the capabilities around SW and HW tracing solutions for functional and performance debug and optimization. While this chapter will focus on it initialization requirements and setup steps in Skylake/Kaby Lake PCH, it will also briefly touch on the Intel TH BIOS usage.

There are multiple output possibilities to choose from for each input trace to go to:

1. DCI
2. System Memory

The output ports are indicated by the GTHOPT0/1 (MTB_BAR + GTH_CSR_BAR + 00h/04h), and the input traces could be mapped to anyone of the output by manipulating the SWDEST[N] or GSWTDEST depending on the input master number, M.

*If M < 256, use SWDEST[M/8] bit [(M&7) <<2] + 3: (M&7) <<2], else, use GSWTDEST bit [3:0]. For Skylake/*Kaby Lake *PCH BIOS, it can use the master that is allocated to the logical processor ring-0 SW, which is from 32-39; thus, the selection for the destination is by programming SWDEST[4].*

Output	Enabling Requirement
DCI	• Program MSCnCTL (CSR_MTB_BAR + A0100h +n*100h) bit[5:4, 0] to [10b, 1]
System Memory	• Memory initialization must be done. • Program MSCnBAR (CSR_MTB_BAR + A0108h +n*100h) to specifies the starting address (shifted left by 12 bits) of the trace buffer • Program MSCnCTL (CSR_MTB_BAR + A0100h +n*100h) to the desired mode and enable it • If CSR-driven mode is used, program MSCnSIZE (CSR_MTB_BAR + A010Ch +n*100h) to specifies the size of the system memory buffer for MSCn in units of 4kB pages

There are multiple inputs possibilities to Intel TH:

1. HW traces
2. SW traces from microcontrollers, including processor

There could be multiple inputs ongoing concurrently; each SW trace input is represented by its master number. Each SW master can support multiple channels and the number of channel is indicated by CHLCNT (CSR_MTB_BAR + 4004h)

BIOS has to be aware that it is not the only agent that configures the Intel TH at early boot. An external Intel TH debugger (host debugger) can be connected to the system through DCI and configure Intel TH through that interface, BIOS must not override it. The host debugger however, still relies on the BIOS to:

1. setup SW_BAR and FW_BAR
2. set MSE
3. setup MSC if memory target is selected
4. set BME

30.1 Intel® TH Initialization

Due to the Intel TH usage for tracing, there could be a need to initialize it early for BIOS trace purpose before its BIOS policy is ready to be used. For such use case, and since the "Host Debugger" is the only mode supported by the BIOS if the Intel TH is to be used, the "Scratchpad0 bit [24]", CSR_MTB_BAR + E0h [24] that would be set when the Intel TH debugger is connected, which in turn would be used as in indication to enable the Intel TH.

To support the potential early call to initialize Intel TH by platform code as well as the Intel TH reference code init routine that is always executed, There would an init library routine created for these 2 callers. The init library routine should include this:

1. Write 00h to HSWPGCR1[7:0], PWRMBASE offset 5D0h [7:0] and then read back to un-power gate the TraceHub (in case it was power gated in earlier boot).
4. Read MSE. If MSE is set, skip the rest.
5. Program CSR_MTB_BAR and enable MSE. Refer to section 5.9.1 for the allocation.
6. Read "Scratchpad0 bit [24]", CSR_MTB_BAR + E0h [24] and input parameter "EnableMode".
   a. If both are zero,
      i. Clear CSR_MTB_BAR and MSE
      ii. Skip the rest.
   b. If this routine is called by the platform code, the EnableMode could be Disable/HostDebuggerMode, though the expectation on the optional platform early call is to pass in the EnableMode as disable, relying on the Intel TH debugger to enable it through the "Scratchpad0 bit [24]". By this practice, it gives the validation team the capability to use single debug BIOS to validate the early trace functionality and code path that enable/disable Intel TH using BIOS policy.
   c. Program GTH_FREQ to 357MHz by set CSR_MTB_BAR + CCh = 15476340h.
2. Clear MSE
3. Program SW_BAR. Refer to section 5.9.1 for the allocation.
4. Program FW_BAR (PCI offset 70h+ 74h) – aka FTMR. Refer to section 5.9.1 for the allocation. There is also a need to program the needed fabric decode per 2.5.2.1.
   a. One CPU related MSR need to be programmed with this BAR value and lock. Refer to CPU BWG for details.
5. Enable MSE and BME
6. If "FW Trace Enable" is enabled, set up destinations for the "Host" Masters (for example BIOS trace usage).
   i. Program SWDEST4, CSR_MTB_BAR + 18h [31:0] to 88888888h

Note: FW_BAR will be given an address range <4GB since there is one agent has that limitation

The later Intel TH reference code must do additional flow below

7. If Intel TH is to be disabled (indicated by its MSE being cleared), power gate the Intel TH.
   a. If DCI is to be enabled (refer to section 13.7), write 80000000h to HSWPGCR1 (PWRMBASE offset 5D0h).

      Else

      Set CSR_MTB_BAR + 2000h[0] to 1.

      Set HSWPGCR1 (PWRMBASE offset 5D0h) bits 0 and 31 to enable Intel TH power gate and lock the power gate control register.
   b. Set PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[TraceHub][8] to 1
   c. Set PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[TraceHub_ACPI][0] to 1 (refer to section 5.3.1)
   d. If Intel TH is to be enabled

      Set CSR_MTB_BAR + 2000h[0] to 1.
8. Lock the power gate control. Write 80000000h to HSWPGCR1 (PWRMBASE offset 5D0h).

30.1.1 Further Initialization Steps for Post Memory

1. On S3 resume, skip the rest
7. Read “MSE, TraceHub PCI offset 4h [1]. If it is 0, skip the rest.
8. On G3/S5/warm boot,
   a. Read Msc0En, CSR_MTB_BAR + A0100h[0]
      i. If it is 1, skip to next step. (in case the external Intel TH debugger tool has started tracing)
      ii. Else, allocate memory according to the Memory Region 0 Buffer Size value from the BIOS policy and program the Msc0Base (CSR_MTB_BAR + A0108h) and Msc0Size (CSR_MTB_BAR + A010Ch) accordingly. These system memory ranges have to be uncached. These ranges of memory should be declared as reserved memory range as well.
   b. Read Msc1En, CSR_MTB_BAR + A0200h[0]
      i. If it is 1, skip to next step.
      ii. Else, allocate memory according to the Memory Region 1 Buffer Size value from the BIOS policy and program the Msc1Base (CSR_MTB_BAR + A0208h) and Msc1Size (CSR_MTB_BAR + A020Ch) accordingly. These system memory ranges have to be uncached. These ranges of memory should be declared as reserved memory range as well.
9. On S4 resume,
   a. Read Msc0Size, CSR_MTB_BAR + A010Ch[31:0]
      i. If it is 0, skip to next step. (to check if earlier boot has allocated memory and S4 path need to have same memory map)
      ii. Else, allocate memory according to the Memory Region 0 Buffer Size value from the BIOS policy and program the Msc0Base (CSR_MTB_BAR + A0108h) and Msc0Size (CSR_MTB_BAR + A010Ch) accordingly. These system memory ranges have to be uncached. These ranges of memory should be declared as reserved memory range as well.
   b. Read Msc1Size, CSR_MTB_BAR + A020Ch[31:0]

i. If it is 0, skip to next step.

ii. Else, allocate memory according to the Memory Region 1 Buffer Size value from the BIOS policy and program the Msc1Base (CSR_MTB_BAR + A0208h) and Msc1Size (CSR_MTB_BAR + A020Ch) accordingly. These system memory ranges have to be uncached. These ranges of memory should be declared as reserved memory range as well.

2. Make sure the memories allocated in earlier step are declared under system resource in ACPI table.
3. To allow BIOS debug trace usage without the needs to read the SW BAR value that could change due to the BIOS PCI enumeration steps
   a. Hide the Intel TH PCI configuration space by setting PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[TraceHub][0] to 1
   b. On PCI enumeration completion or later phase, unhide the Intel TH by setting PCR[PSF_3] + "AGNT_T0_SHDW_CFG_DIS"[TraceHub][0] to 0. This is to allow OS SW to discover the Intel TH for its usage. (Note: If the Intel TH was disabled, this step will not reveal it because the it was disabled using the PSF function disable, which is a superset)

30.2 BIOS Debug Trace Usage

It is recommended to add debug messages support for BIOS Debug Trace log in release BIOS (Port 80 Post code messages) and debug BIOS (Port 80 Post code, PEI, ACPI, DXE SMM messages).

Since the first set of software masters are reserved for Ring0 IA/Core trace, these will be used for BIOS trace as well. Read STHCAP0 (CSR_MTB_BAR + 4000h [15:0]) to get SW_MSTR_START or hardcoded as 32 for Skylake/Kaby Lake PCH. Read STHCAP1 (CSR_MTB_BAR + 4004h) to get CHLCNT or hardcoded as 80h for Skylake/Kaby Lake PCH.

The BIOS can make use of the master 32-39 depending on the executing logical processor, and channel 13-22. The master number 32, channel number 0 would be an offset 0h from SW_BAR. On BIOS event that is serialized, or ACPI code, it is expected a fixed master and channel assignment could be used.

The address for the software trace channel is calculated as follows:

```
UINT64 TraceHub_addr(UINT8 master_num, channel_num){

    UINT64 TraceAddr = SW_BAR +  40h*( CHLCNT * master_num - SW_MSTR_START) +
    40h*channel_num;
    return TraceAddr;
    }
```

Function pseudo code (for reference) is as follows:

```
Void TraceHub_dbgstr(char* string, uint8 subtype, uint8 master_num, unit8 channel_num){

    UINT64 TraceAddr, header, data;
    UINT32 length, aligned_length, x;

    TraceAddr = trace_addr(master_num, channel_num);
    header = (type << 10) | subtype; //type = 2 (debug string)
    *(uint64_t*)(TraceAddr + 18h) = header; //this generates a DxM_TS
```

```
    data = 0;
    length = strlen(string);
    aligned_length = length; //we need an 8-char aligned size (since we are going to write 8
chars at a time)

    if (length % 8 > 0) {
        aligned_length += 8 - (length % 8);
    }

    for(x = 0; x < aligned_length; x++)
        data = data << 8; //get ready to shift in next char
        if (x < length) {
            data |= string[x];
        }
        else { // need to pad with because (x >= length)
            data |= "\0"
        }

        if ((x+1) % 8 == 0) { //write each group of 8 chars
            *(uint64_t*)(TradeAddr + 00h) = data
        }
    }

    *(uint64_t*)(TraceAddr + 30h) = 0h; //this generates a FLAG (flag ignores data)
}
```

§ §

31 Lewisburg PCH

Lewisburg is the name of the EP/EX server/workstation market segment PCH Platform

Controller Hub. The BIOS related information for Lewisburg has much in common with Skylake/Kaby Lake PCH-H due to shared interfaces between the two devices. This chapter is intended to document the differences between the Skylake/Kaby Lake PCH-H and Lewisburg PCH chipsets. All the previous sections from this BIOS Specification apply to both Skylake/Kaby Lake PCH-H and Lewisburg unless noted in this chapter. This chapter is structured to contain sections and subsections which correspond to each one of the previous chapter of the BIOS Specification. BIOS specific changes that have been identified for the Lewisburg device added features over Skylake/Kaby Lake PCH-H are limited at present. Document additions for Lewisburg features and other new guidance will be added in as they become available. Until details are finalized these are labelled as "To Be Determined" **(TBD)**.

Note: If Skylake/Kaby Lake PCH-H has a programming requirement exclusively for certain engineering steppings, that programming requirement will also only apply to certain engineering steppings of Lewisburg. Additional information is TBD.

31.1 Introduction

Additional programming guidance is **TBD.**

The Chapter 1 contains the basic information for Skylake/Kaby Lake PCH-H which applies for Lewisburg as well.

The Lewisburg has, at the high level, two versions: LBG-SD and LBG-NS (small die and Network Services – final labeling is **TBD**). Small die is intended to be used in standard servers and is an extension to Skylake/Kaby Lake PCH-H with some devices present and removed. NS is intended in solutions where high networking processing power is needed.

The NS version is a very new approach in server PCH design. It allows Intel to be more flexible in filling customer needs and creates many new use cases and gives additional flexibility in designing server systems. The customer may for example choose the LBG as separate device on separate PCIe card to use only Network Services functionality. Please refer to EDS for details.

In this document BIOS implications for both versions will be described.

31.2 Intel Platform Controller Hub (PCH) Overview

Additional programming guidance is ***TBD.***

The Chapter 2 contains overview of the Skylake/Kaby Lake PCH-H, although there are some differences:

- No Display Support.
- No Integrated Sensor Hub
- No Intel Serial I/O

- USB xHCI controller has 14 ports instead of 12
- Some ICC changes (frequencies, etc.)
- MS Unit and server-SATA (sSATA) added with two Manageability ROMs, 3 SMBus controllers, additional 6 SATA ports
- Innovation Engine added
- Additional MCTP configuration
- Some SVID and SID changes of PCH Devices
- Some Reliability, availability and serviceability (RAS) changes
- Some changes in Reset handling - Demoted Warm Reset (DWR)

In Lewisburg NS version the component called Acceleration Complex is added with Content Processing Modules and two Dual 10G Ethernet controllers and with Uplink ports to connect to the Host CPU (aside of DMI).

31.2.1 To 2.2 Primary to Sideband Bridge (P2SB)

UEFI FW shall configure P2SB to block access to all Lewisburg fabrics (PSF1-10).

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.2.1.1 To 2.2.1 SBI Programming

UEFI FW cannot set Hardware Autonomous Enable (HAE) and PMC Power Gating Enable (PMCPG_EN):

1. Write P2SB PCI offset E4h[5] = 0b
2. Write P2SB PCI offset E4h[0] = 0b

31.2.1.2 To 2.2.3 Endpoint Mask:

1. Skip setting P2SB PCI offset C4h [29, 28, 27, 26]

instead:

2. Mask PSF fabrics to block access from Host:
   a. If P2SB PCI offset 0h quals FFFFh, meaning device is not present
      i. Set P2SB PCI offset E0h+1 = 0, to reveal P2SB device
   b. For each PSF 1-10
      i. Set EPMASK0 P2SB PCI offset 220h + PSFX_EPMASK[PSFX_BIT] = 1b
3. Lock the EPMASK registers:
   a. Set EPMASK0 P2SB PCI offset E0h+2[1] = 1b

31.2.2 To 2.5.3 LPC Cycle Decoding

Skip configuring LPC (enhance port 8xh LPC decoding) on Early PEI phase.

31.2.3 SVID and SID of PCH Devices

Remove entries from SVID / SID table:

1. Serial IO UART0 Controller Registers (D30:F0)
2. Serial IO UART1 Controller Registers (D30:F1)
3. Serial IO SPI0 Controller Registers (D30:F2)
4. Serial IO SPI1 Controller Registers (D30:F3)
5. SCS eMMC Controller Registers (D30:F4)
6. SCS SDIO Controller Registers (D30:F5)
7. SCS SDCARD Controller Registers (D30:F6)
8. CIO2 Registers (D20:F3)
9. ISH Controller Registers (D19:F0)
10. Serial IO I2C0 Controller Registers (D21:F0)
11. Serial IO I2C1 Controller Registers (D21:F1)
12. Serial IO I2C2 Controller Registers (D21:F2)
13. Serial IO UART2 Controller Registers (D25:F0)
14. Serial IO I2C5 Controller Registers (D25:F1)
15. Serial IO I2C4 Controller Registers (D25:F2)

Add entries to SVID / SID table:

1. IE HECI1 Controller Registers (D16:F0)
2. IE HECI2 Controller Registers (D16:F1)
3. IE IDER Controller Registers (D16:F2)
4. IE KT Controller Registers (D16:F3)

31.3 Flash Programming Considerations

Flash component is accessible in the same way as in Skylake/Kaby Lake PCH-H. Additionally in LBG the Flash needs to have additional regions for:

- IE Firmware
- In Lewisburg NS version 2 OpROMs (EFI Drivers) for two Dual 10G Ethernet Controllers

31.4 Reset Control

Same as Skylake/Kaby Lake PCH-H PCH.

For Lewisburg NS version the reset flows has been extended to handle the Reset Unit a new block in Acceleration Complex. There is no BIOS impact regarding AC reset and only soft straps changes might be considered for different SKUs.

31.4.1 Demoted Warm Reset

A Demoted Warm Reset or Dirty Warm Reset (DWR) is a warm reset that needs to dump errors after a Reset. This procedure will skip all handshakes including Go_S1 between PCH and CPU during a normal warm reset flow, causing a dirty warm reset.

Enable this feature in the register GBL2HOST_EN[0] bit G2H_FEAT_EN. After a DWR occurs, when booting, BIOS must read register HPR_CAUSE0[15] bit GBL_TO_HOST in PCH to determine if it was a DWR. DWR are not reliable and BIOS should not boot to OS, instead BIOS is expected to perform minimal initialization steps, read error logs, deposit error lost to non-volatile memory and issue a global reset. BIOS should attempt to collect error logs as early in the boot phase as possible such as right after System BSP selection.

Please refer to supporting documents for more details:

- DWR changes-sept14-r00.pptx
- DWR_recipe_changes.pdf

A routine PchIsDwrFlow() is used in all modules for DWR detection.

Two GUIDS are implemented:

- Definition for DirtyWarmResetPpi including ExecuteDirtyWarmResetFlow function (internally implemented in PCIe Root Ports DirtyWarmResetExecute):
    - **gDirtyWarmResetGuid = { 0xe60fe263, 0xac2b, 0x43d6, { 0xb3, 0xc7, 0x0d, 0x9d, 0xdc, 0x5a, 0x99, 0x1c }}**

- Signal Ppi installed upon function described above, allowing to register callback for this event
    - **gDirtyWarmResetSignalGuid = { 0x24b9a592, 0x4cfc, 0x4c8f, { 0x86, 0xf4, 0x87, 0x28, 0x2d, 0x7f, 0x9e, 0x9c }}**

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

For Gbe related programming check: 31.10 Gigabit Ethernet Controller.

For MCTP related programming check: 31.31 Management Component Transport Protocol (MCTP)

For DWR flow**:**

- skip PchPrintPolicyPpi when Installing PCH Policy
- skip PrintPchPciConfigSpace
- skip PchProgramSvidSid - Program Pch devices Subsystem Vendor Identifier (SVID) and Subsystem Identifier (SID).
- Skip PchMiscInit - Internal function performing miscellaneous init needed in early PEI phase

More details from the Code insights:

1. Pch setup menu DirtyWarmReset item added in PCH Configuration form.

2. Pch policy has been extended to have a way to configure the RC Code reading BIOS menu item.

3. PCH register definition headers for PMC and LPC have extended with new registers to fulfill the requested registers setting.

4. During early PchInitPreMem phase we detect whether the reset has been promoted to Dirty Warm Reset. If so we do a global reset to the customers can stop and read registers using ITP debugger.

5. The DWR functionality enabling was added to PchPm.c module of the PeiPchInitLib private library. The enabling code checks whether we don't run on HEDT/WS LBG SKU and then checks for menu item whether the DWR is enabled and then programs the feature registers in PMC device.

31.4.2 PEI Phase related flow

For pre-memory initialization phase:

- Install DWR PPI gDirtyWarmResetPpi
- Log internal PSF errors

31.4.3 PCIe Root Port related flow

In order to execute Dirty Warm Reset:

1. If (PchIsDwrFlow() == FALSE) finish programming,
   else check whether previous reset was promoted to Dirty Warm Reset.
   a. In order to determine DWR flow check if Host partition reset causes indicate that global reset converted to host reset was detected: PWRMBASE + 12Ch [15] is 1b
2. Get available end point device from PCIe Root Port base address:
   a. On error, report: 'can't get end point device available'
   b. On success:
      i. Leave BMC PCI device available
      ii. A config write is required in order for the device to re-capture the Bus number, according to PCI Express Base Specification, 2.2.6.2 Write to a read-only register VendorID to not cause any side effects.
3. Add debug ability to stall in the DWR flow prior to the reset
4. Signal DWR guid by installing gDirtyWarmResetSignalPpi
5. If BIT25 == 1 => BIOS is requested to stall for DWR debug/validation
   Set bit26 to indicate we are in stalled flow
6. Stall until 3rd party agent clears BIT26 indicating ready to reset
1. If reset was requested (could locate gPchResetPpiGuid)
   a. Issue GlobalResetWithEc

31.4.4 Power Management related flow

1. If PchPolicy->PmConfig.DirtyWarmReset
   a. MmioWrite32 (PchPwrmBase + R_PCH_PWRM_GBL2HOST_EN, PchPolicy->PmConfig.PchGbl2HostEn.Value);
2. if (PchPolicy->PmConfig.Dwr_MeResetPrepDone) {
   a. MmioOr32(PciPmcRegBase + R_PCH_PMC_ETR3, BIT17);
      } else {
      MmioAnd32(PciPmcRegBase + R_PCH_PMC_ETR3, (UINT32)~BIT17);
      }
3. if (PchPolicy->PmConfig.Dwr_IeResetPrepDone)
   a. MmioOr32(PciPmcRegBase + R_PCH_PMC_ETR3, BIT16);
   b. } else {
   c. MmioAnd32(PciPmcRegBase + R_PCH_PMC_ETR3, (UINT32)~BIT16);
   d. }
4. if (!IS_PCH_LBG_WS_LPC_DEVICE_ID(DeviceId)) {
   a. Set G2H_EN.G2H_FEAT_EN enable bit in the PCH during each boot
5. if (PchPolicy->PmConfig.StallDirtyWarmReset){

31.5 Basic BIOS Requirements

Same as Skylake/Kaby Lake PCH-H PCH.

31.6 Interrupt Support on Platform Controller Hub (PCH)

Configuration for devices such as Intel Serial I/O, SCS, ISH and OTG devices is not supported in Lewisburg.

The number of RTEs has been extended, however for Lewisburg there is no identified interrupts that might need to use this many entries. Therefore for Lewisburg we disable the extension entries 24-119 by default.

To enable extended RTE number, refer to chapter 6, Configuration of the Platform Controller Hub (PCH) IOxAPIC.

Additionally for Lewisburg some interrupts mapping needs to be done for new Lewisburg devices. Steps are LBG specific devices which use INTx->PIRQy mapping are: server-SATA (sSATA) and Innovation Engine HECI's..

31.7 Direct Media Interface (DMI) and On-Package Direct Media Interface (OP-DMI)

Same as Skylake/Kaby Lake PCH-H PCH.

Note: The only supported DMI link width for Lewisburg is x4. (4926615)

31.8 PCI Express Support

The support is the same as for Skylake/Kaby Lake PCH-H as Lewisburg uses the same HW blocks (IP). The new approach in Skylake/Kaby Lake PCH-H and in Lewisburg is to connect all the Modular PHYs to Flexi I/O Adapter which then switches particular PHYs into specific controllers (PCIe, SATA, GbE, xHCI) using softstrap configuration. While in Skylake/Kaby Lake PCH-H, one Flexi I/O adapter is used; in Lewisburg we have two (WM20 and WM26), and all the PCIe configuration is being done using IOSF-SB interface of WM26 which has different Port ID. Therefore all the steps for BIOS described for Skylake/Kaby Lake PCH-H are valid. However, for Lewisburg the BIOS must use WM26's port ID to program that.

The Lewisburg introduces different PHY muxing combinations then Skylake/Kaby Lake PCH-H.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.8.1 To 8.15 Additional PCI Express* Programming Steps

UEFI FW needs to perform Power Management-related programming:

8. Skip clock gating programing
    b. Leave to default value: Dxx:Fn + E1h [1, 0]
    c. Leave to default value: Dxx:Fn + F7h [6]
12. Skip dedicated clkreq per port related programming
    a. Skip program Dxx:Fn + 424h [7, 6, 5, 4]

31.8.2 To 8.2.1.1 PCIe* Function Disable Handling

Skip some power management steps needed along with function disable programming above:

3. Leave to default value: PCI offset 420h[31]
4. Leave to default value: PCI offset 424h[8]

31.8.3 To 8.15.1 Power Optimizer Configuration

1. Enable support Latency Tolerance Reporting (LTR)
    a. Based on the flow diagram shown in Figure 8-8, BIOS will determine the override value (snoop and/or non-snoop) to be programmed in B0:Dxx:Fn + 400h
        iv. Leave to default value B0:Dxx:Fn + D0h [13:12]

31.8.4 End of PEI

UEFI FW needs to set link width (PchSetLinkWidth) according to policy.

31.8.5 DWR flow

Skip configuring PciERWORegInit Initialize R/WO Registers (System BIOS must read the register and write the same value back to the register) on DWR flow (PchIsDwrFlow()).

31.8.6 Maximum Payload Size

Maximum Payload Size can also be configured in the Device Capabilities register to be either 128B or 256B regardless of the capability of the device attached during boot.

31.9 Intel® High Definition Audio Controller

Same as Skylake/Kaby Lake PCH-H PCH.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.9.1 To 9.6.6 Power Management and Clock Gating Requirement

2. Leave to default PCR[DSP] + 530h[5] to disable power gating

31.10 Gigabit Ethernet Controller

Same as Skylake/Kaby Lake PCH-H PCH.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.10.1 To 10.8.1 LTR Programming

The System BIOS must program

- the maximum snoop/non-snoop platform latency values to 08280828h in the GbE controller's PCI LTR capability register at GbE PCI offset 0A8h.

Controller disabling

Disable GBE if PCH soft strap R_PCH_SPI_SS119_PMC 0x1DC [14] is enabled:

- Set PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[GBE][8] to 1, to perform PSF T0 function disabling

31.10.2 DXE phase flow

Skip code checking if Gbe is present while on DWR flow (PchIsDwrFlow()).

31.11 SMBus

Same as Skylake/Kaby Lake PCH-H PCH.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.11.1 Disabling SMBus

While disabling Smbus add PSF programming:

1. Set PCR[PSF_2] + "AGNT_T0_SHDW_PCIEN"[SMB][8] to 1

31.11.2 Post-mem init

TO disable HW Power Gating:

2. Clear PCE.HAE, PCR[SMB] + 10h [5] = 0b.
3. Clear PCE.HAE, PCR[SMB] + 10h [0] = 0b.

31.12 Extensible Device Controller Interface (xDCI)

xDCI is not supported on Lewisburg, nevertheless some programming is required for proper disabling.

The USB OnTheGo (OTG) is not supported.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.12.1 Pre-mem

UEFI FW shall additionaly run xDCI initialization at pre-mem phase.

31.12.2 To 12.1 xDCI Enabling and Disabling

To disable xDCI:

3. Wait for host owned (DUAL_ROLE_CFG, xHCI MBAR + 80DCh bit [29]) SS DRD mode to be 1.

    a. Set xDCIBAR + 80D8h [1,0] to 01b, to statically assign USB3 port 1 and USB2 port 1 to xHCI for disabled OTG

9. Skip seting xDCI PCI offset A2h [2, 1]

31.13 XHCI Support

Same as Skylake/Kaby Lake PCH-H PCH.

Programming steps are the same as for Skylake/Kaby Lake PCH-H for each USB3 capable port. However there are some differences that need to be kept in mind:

- Lewisburg has 14 ports. Iterating on the USB ports in the BIOS must be aware of that.
- Lewisburg has different Flexi I/O muxing combinations where certain USB3.0 ports can be routed to different Modular PHYs.
- Possible programming steps different for Lewisburg are TBD.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.13.1 To 13.4.1 xHCI Controller Setup

3. Skip program xHCI PCI offset 50h = 0FCE6E5Fh
5. Skip program xHCI PCI offset 74h[1,0] = 1b, 1b

Skip Delay 20us.

6. Skip program xHCI PCI offset 74h[1,0] = 0b, 0b
9. Skip program xHCI PCI offset 50h = 0FCE6E5Fh
10. Skip program xHCI PCI offset A2h[3, 2] to 0b, 1b
16. Set xHCI MBAR + 80A4h[30, 27, 24, 23, 16, 15, 13, 11, 10, 9, 8, 3, 2] to 1b, 1b, 1b, 1b, 0b, 0b, 0b, 0b, 0b, 0b, 0b, 1b, 1b
17. Set xHCI MBAR + 80A8h[24] to 1b for LBG A0 Stepping and [27, 26] to 1b, 1b for other LBG Steppings.
18. Set xHCI MBAR + 80E0h[22, ~~16,~~ 9, 6] to 0b, ~~0b,~~ 0b, 1b
22. Set xHCI MBAR + 816Ch[19, 18, 17, 16, 14, 13, 12, 11, 10, 9, 8, 4, 5, 3, 2] to 1b, 1b, 1b, 1b, 1b, 0b, 0b, 0b, 0b, 0b, 0b, 1b, 0b, 1b, 1b
23. Set xHCI MBAR + 8140h = FF08003Ch for LBG steppings.
24. Set xHCI MBAR + 81C4h [11, 8, 3 ,2, 1, 0] to 1b, 1b, 0b, 0b, 0b, 0b
28. Set xHCI MBAR + 80B4h[5, 2] to 1b, 1b – for LBG steppings.
31. Set xHCI MBAR + 8110h[20,11,2] to 1b, 1b, 0b
    a. Set xHCI MBAR + 8128h[47:36] to FFFh

38.Set xHCI MBAR + 818Ch[16, 8] to 1b, 0b

    b. Set xHCI MBAR + 8190h[22] to 0b for LBG A0 Stepping.

40. Set xHCI MBAR + 819Ch~~[20] 1b~~, [16] 1b, [14] 1b, [13] 1b, [12] 1b, [11:8] 3h, [7:4] 7h, [3:0] Dh

UEFI FW must perform USB 2.0 high-speed signals quality tuning by perform below steps through sideband interface access.

2. Set PCR[USB2] + 4001h[25:24] = 11b
   a. Set PCR[USB2] + 7000h[31:24] to 50h, [23:16] to 50h
6. Set PCR[USB2] + 7F04h[15] to 1b, [10:7] to 1001b
7. Set PCR[USB2] + 7028h[31:0] = 04022C0Ch
8. Set PCR[USB2] + 7F03h[31:0] = 0B000B00h

Aditionally per port UEFI FW must perform USB 2.0 high-speed signals quality tuning by perform below steps through sideband interface access:

1. Set PCR[USB2] + 4000h + port_num * 100h [31, 30, 29, 28, 27, 26, 4] = 1111111b

After all Skylake/Kaby Lake PCH-H PCH programming, additionally:

1. Set xHCI MBAR + 80E0h[16] to 0b
2. To disable power gating program xHCI PCI offset 50h = 07CE6E5Fh (bit27 cleared compared to 0FCE6E5Fh)

31.13.2 To 13.4.2. Super Speed Port Enable

When setting Super Speed Ports terminations on enabled ports (80B8h) before writing preserve the bits already set in this register apart from bits 31, 30 and 29.

As XHC Save/Restore function will be broken if reset during S3 resume, before exiting set 8090h[10] to 0b.

31.13.3 To 13.7 DCI Support

On Pch Policy installed (as DPG is not POR for LBG):

1. Skip settng PCR[DCI] + 30h bit[5], and clearing 30h [2,1,0]

31.13.4 Locking xHCI Register Settings

Note: Before setting OC Configuration Done bit, configure U20CM (XHCI PCI offset B0h-CFh) and U3OCM (XHCI PCI offset D0h-EFh) USB Overcurrent Pin Mapping registers.

31.14 Integrated SATA Controllers

The Platform Controller Hub (PCH) Serial ATA (SATA) controller supports a maximum of 14 SATA ports. See EDS for details on number of ports and SATA speed support per SKU.

An additional controller is available and is referred to as the sSATA controller on Lewisburg.

For sSATA programming steps will described in chapter 31.31.1

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.14.1 To 14.1 SATA Controller Initialization

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 or no lanes assigned to SATA when in AHCI mode, perform disabling steps as indicated in section 14.3 and exit here
2. Perform the programming requirement steps as indicated in section 14.1.6.
3. For LBG always set ASSEL to 110b (512k).

31.14.2 To 14.1.3 Initialize Registers in AHCI Memory-Mapped Space

- Set EMS (ABAR + 00h[6]) if enclosure support is enabled; this is for enable Enclosure Management capability.

31.14.3 To 14.1.5 Power Optimizer Configuration

For Skylake/Kaby Lake PCH-H:

6. Set SATA SIR Index D0h[15:0] = 8828h

12. Set SATA SIR Index A8h[19:16] = 0000b to disable dynamic power gating

31.14.4 To 14.1.6 Additional Programming Requirements during SATA Initialization

For Skylake/Kaby Lake PCH-H:

1. System BIOS must set SATA SIR Index 98h[8:0] = 183h as part of the chipset initialization prior to SATA configuration
   a. Server must Program SATA SIR Index 98h[15] to 1b

5. Program SATA SIR Index 9Ch[29, 23, 20, 19, 18, 12:7, 6:5, 3:2] to [1, 1, 1, 1, 1, 4, 1, 1]
   b. Note: bit 22 is optional for server platform

   5.1. Program SATA SIR Index 9Ch[30] to 1b to disable Trunk Clock Gating

31.14.5 To 14.3 SATA Controller Disabling

For Skylake/Kaby Lake PCH-H:

5. Enable all SATA dynamic clock gating and dynamic power gating features
   a. SATA SIR offset 9Ch bit [29, 23] = [1, 1]
   b. SATA PTM2, “SIR offset 84h” bit [23:16] to [FFh]
   c. SATA PTM4, “SIR offset 8Ch” bit [23:16, 7:0] to [FFh, FFh]
   d. SATA CTM3, “SIR offset A0h” bit [15] to 1
   e. SATA CTM4, “SIR offset A4h” bit [26, 25, 24, 14, 5] to [0, 0, 0, 1, 1]
   f. Skip programming SATA CTM5, “SIR offset A8h” bit [17, 16]
   g. For each RST PCIe Storage Cycle Router, program the "Extended General Configuration Register" field, PCI offset 354h[20, 19, ~~17,~~ 16] = 1b, 1b, ~~1b,~~ 1b

31.14.6 Additionally for LBG:

1. Set PWRMBSE + 628h [22] to 1b to disable Sata Controller in PMC
2. Set PWRMBSE + 628h [27] to 1b to disable sSata Controller in PMC

31.15 High Performance Event Timer (HPET) Support

Same as Skylake/Kaby Lake PCH-H PCH.

31.16 GPIO Handling

Please refer to the EDS document for register layout details.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.16.1 To 16.4 GPIO Programming Steps

UEFI FW should default all the 3 pins listed to GPIO Input and remove any possible selection of native function in the programming: GPP_B_2, GPD_7, GPD_9.

UEFI FW should skip handling requirement (leaving it in GPIO mode) on gSPIx_CSB pins.

31.16.2 To 16.8 GPIO Pad Special Handling.

Skip 16.8.1 PCH-LP GPP_A_12.

Ad 31.22 Intel Serial I/O Bus

Skip setting SerialIo I2C controller pins into native mode, as not supported on Lewisburg (see 31.22).

Ad 31.25 Intel Sensor Hub

As not supported on Lewisburg:

Skip setting ISH I2C controller pins into native mode,

Skip setting ISH UART controller pins into native mode.

Skip setting ISH SPI controller pins into native mode.

Skip setting ISH GP pins into native mode

31.17 Thermal Management Support

Thermal Device is handled by BIOS the same way as in Skylake/Kaby Lake PCH-H, although some thresholds and levels need to be adjusted accordingly in Platform Policy.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.17.1 To 17.5 Thermal Throttling

Program sSATA Indexed Register Index (SIRI) and Data exactly the same as for SATA.

31.18 Integrated Clock Support

No BIOS changes for Lewisburg. Same as Skylake/Kaby Lake PCH-H.

31.19 Power Management

Additional programming guidance is ***TBD.***

The steps are the same as for Skylake/Kaby Lake PCH-H.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.19.1 To 19.5 Additional Power Management Programming

- Set PWRMBASE + offset 48h = 020DDF01h.
- Set PWRMBASE + offset 4Ch = 00010001h.
- Set PWRMBASE + offset 50h = 020DDF01h.
- Set PWRMBASE + offset 54h = 00010001h.
- Set PWRMBASE + offset 58h = 0001C000h.
- Set PWRMBASE + offset 68h = 04041C00h.
- Set PWRMBASE + offset 80h = 9F85DB01h.
- Set PWRMBASE + offset 84h = 0181C7E1h.
- Set PWRMBASE + offset 88h = 9F85DB01h.
- Set PWRMBASE + offset 8Ch = 0181C7E1h.

- Set PWRMBASE + offset 98h = 0001C000h.
- Set PWRMBASE + offset A8h = 00181300h.
- Set PWRMBASE + offset ACh = 02006001h.
- Set PWRMBASE + offset B0h = 00181300h.
- Set PWRMBASE + offset B4h = 02006001h.
- Set PWRMBASE + offset C0h = 00060A00h.
- PWRMBASE + offset C8h [11:8] are set based on board config.
- Set PWRMBASE + offset D0h = 00200840h.
- Set PWRMBASE + offset D4h = 0000F026h.
- Set PWRMBASE + offset E0h = 00040001h.
- Set PWRMBASE + offset E4h = 16BF0003h.
- Set PWRMBASE + offset F4h = 0000F000h.
- Skip setting PWRMBASE + offset 0E8h [30] to avoid enabling USB2 PHY SUS Well Power Gating.

Configure Miscellaneous Power Management:

- Set t581 timer to 10ms:
  - Set PWRMBASE + offset 18h [1:0] = 11b
- Enable the Global to Host reset promotion feature by setting 10Ch[0]=1 when setup option is Enabled, for HEDT SKUs it needs to be disabled.
  - Set PWRMBASE + offset 10Ch [0] = 1b
  - If PmConfig.ME Reset Prep Done
    - Set PMC PCI offset ACh [17] = 1b else set to 0b
  - If PmConfig.IE Reset Prep Done:
    - Set PMC PCI offset ACh [16] = 1b else set to 0b
  - If not LBG_WS_LPC SKU
    - Set PWRMBASE + offset 10Ch [0] = 1b
  - If PmConfig.Stall during DWR:
    - If BiosSspad7Address (scratchpad7) is not FFFFFFFFh:
      - Set BiosSspad7Address [25] = 1b

Configure Psf Power Management:

- Trunk clock gating for PSF's 1,2,3,8,9 and 10 when with on LBG-NS must be disabled:
  - Program PCR[PSF_1,2,3,4,8,9 and 10] + 4000h [4] = 1b

31.19.2 To 19.15 SPI/eSPI Power Management Programming

1. Set PCR[ESPISPI] + C004h[29,10,4:2,0] = 1b, 1b, 111b,1b to enable dynamic clock gating.

31.20 TCO Support

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.20.1 To 20.4.1. Additional SMI handling of NEW CENTURY of Skylake/Kaby Lake

For LBG Skip additional requirement to clear the SMI status of NEW CENTURY in Skylake/Kaby Lake.

1. Detect NEWCENTURY_STS, if set
    a. Clear PCE.HAE, PCR[SMB] + 10h [5] = 0b
    b. Loop to write clear NEWCENTURY_STS and check it till it's cleared.
    c. Clear TCO_STS.
    d. Set PCE.HAE, PCR[SMB] + 10h [5] = 1b

31.21 Legacy Free Systems

No BIOS changes for Lewisburg. Same as Skylake/Kaby Lake PCH-H.

31.22 Intel Serial I/O Bus

The Intel Serial I/O Bus (LPSS) is not available on Lewisburg. All devices in this section should be treated as Reserved on Lewisburg.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below

31.22.1 To 22.2 Disabling Intel® Serial I/O Controllers

For Lewsiburg skip:

- Performing PSF disable for SPI0-1, UART0-2, I2C0-5 in not fused
- D3 programming for I2C4-5
- PMC PG static bit PWRMBASE + 624h

31.22.2 End of PEI

Skip configuring SIO at S3resume.

31.23 Intel® Smart Sound Technology

No BIOS changes for Lewisburg. Same as Skylake/Kaby Lake PCH-H.

31.24 Intel® RST for PCIe Storage

Programing and BIOS configuration steps are the same as for Skylake/Kaby Lake PCH-H. The only thing that must be kept in mind in the BIOS code for Lewisburg is that it supports PCIe NVM device for remapping up to 3 devices instead of 2.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

Lewisburg has enabled NVM Remapping on all Root Ports.

31.24.1 To 24.3.2. Initialization

The following describes LBG specific changes to the initialization flow for how BIOS detects and configures the NAND Cycle Router.

3. The BIOS does a mini PCIe enumeration or manual selection to search for the PCIe NAND controller that exports an AHCI/NVMe controller function.

   b. Refer to the table below to determine PCIe root port that supports remapping

Bit	LBG
[4:1]	RP1-RP4
[8:5]	RP5-RP8
[12:9]	RP9-RP12
[16:13]	RP13-RP16
[20:17]	RP17-RP20

31.25 Intel Sensor Hub

The Intel Sensor Hub (ISH) is not available on Lewisburg. All devices in this section should be treated as Reserved on Lewisburg.

31.25.1 To 25.1 ISH Disable/Enable Handling

For Lewisburg skip PSF disabling if ISH fuse disabled.

31.26 Camera I/O Controller (LP)

The Camera I/O Controller is not available on Lewisburg and Skylake/Kaby Lake PCH-H. All devices in this section should be treated as Reserved on Lewisburg.

31.26.1 To 26.2 CIO2 Controller Disabling

For Lewsiburg skip PSF disabling if CIO fuse disabled.

31.27 eSPI

Lewisburg RC implements support for eSPI second device (CS1#), two eSPI Slave devices in general. To enable this capability, 'Get' and 'Set' functions for LPC/eSPI generic IO range list and LPC/eSPI memory range decoding were extended to handle two slave devices.

Below address ranges may be considered duplicate with LBG EDS: "17 Enhanced Serial Peripheral Interface".

Address ranges for eSPI second device:

- IOSF SB: Second eSPI Generic Memory Range: PCR [DMI] + 27C0h
- IOSF SB: Second eSPI Generic I/O Range: PCR [DMI] + 27BCh
- PCIe CFG: eSPI CS1# Generic I/O Range #1 (PCCS1GIR1) - Offset A4h
- PCIe CFG: eSPI CS1# Generic Memory Range (PCCS1GMR1) - Offset A8h

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.27.1 To 2.5.1.2 LPC Generic I/O Range Decoding

If Slave Device is "LPC_ESPI_FIRST_SLAVE":

- "Setup "LPC Generic I/O Range #", PCR [DMI] + 2730h~273Fh to the same value program in LPC/eSPI PCI offset 84h~93h

Else:

- "Setup "LPC Generic I/O Range #", PCR [DMI] + 27BCh to the same value program in LPC/eSPI PCI offset A4h
  (Program just one range for eSPI Second Device Generic I/O, instead of range of 4)

31.27.2 To 2.5.1.3 LPC Memory Range Decoding

If Slave Device is "LPC_ESPI_FIRST_SLAVE":

- Setup "LPC Memory Range", PCR[DMI] + 2740h to the same value program in LPC/eSPI PCI offset 98h.

Else:

- Setup “LPC Memory Range”, PCR[DMI] + 27C0h to the same value program in LPC/eSPI PCI offset A8h.

31.27.3 To 2.5.1.6 LPC IO Enables Decoding

If Slave Device is “LPC_ESPI_FIRST_SLAVE”:

- Setup “LPC I/O Enables”, PCR[DMI] + 2774h [15:0] to the same value program in LPC/eSPI PCI offset 82h.

Else:

- Setup “LPC I/O Enables”, PCR[DMI] + 2774h [15:0] to the same value program in LPC/eSPI PCI offset A0h.

For Power Management programming check 31.19 Power Management.

31.28 Storage and Communication Subsystem

The Storage and Communication Subsystem (SCS) is not available on Lewisburg. All devices in this section should be treated as Reserved on Lewisburg.

31.29 HSIO and FIA

All the ModPHY values are being loaded in the same way as in Skylake/Kaby Lake PCH-H. For Lewisburg we will need well defined values for HSIO interfaces.

Lewisburg has two Flexible I/O Adapters. First called FIA0 connects Uplink port x16 and DMI x4 to the Modular PHYs. All other HSIO interfaces like PCIe, Uplink port x8, USB3, SATA are switched by the second FIA called FIA1.

The BIOS code for Lewisburg must ensure all the FIA accesses on IOSF-SB are targeting the proper FIA. Additionally, the BIOS code must ensure all the registers supported by the FIA in Skylake/Kaby Lake PCH-H are also supported in the two FIAs in Lewisburg and have the same meaning.

(IP). The new approach in Skylake/Kaby Lake PCH-H and in Lewisburg is to connect all the Modular PHYs to Flexi I/O Adapter which then switches particular PHYs into specific controllers (PCIe, SATA, GbE, xHCI) using softstrap configuration. While in Skylake/Kaby Lake PCH-H, one Flexi I/O adapter is used; in Lewisburg we have two (WM20 and WM26),

PchGetGbeLaneNum not used anywhere in Lewisburg.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.29.1 To 29 HSIO and FIA

Program Hsio Rx/Tx Equalization according to Lewisburg port number and policy, including:

- PTSS programming for WM20 FIA PCIe Lanes.

Use WM26 PID (PID_FIAWM26 13h) instead of Skylake/Kabylake (PID_FIA CFh) during Common Hsio programming:

- While iterating through the HSIO table for WM26 entries, include Lewisburg PIDs (PID_MODPHY4 EBh and PID_MODPHY5 10h)

While programming the Pcie Pll SSC registers in WM20 consider Lewisburg clocking modes to choose between Wm20Gen2Pll (native and hybrid clock) and SccAlternate (alternate clock).

PCIe lane index to HSIO lane number mapping: 6, 10, 14, 18, 22.

While getting lane info use WM26 and PortIds (PID_MODPHY1 E9h, PID_MODPHY2 A9h), also implement additional one for WM20 (PID_MODPHY4 EBh and PID_MODPHY5 10h).

Below can be considered duplicate from EDS:

Private Configuration Space Register Target Port IDs:

- MODPHY1 (HSIO Lanes #0 - 9) 0xE9 (Gen2 PLL)
- MODPHY2 (HSIO Lanes #10 - #25) 0xA9 (Gen3 PLL)
- MODPHY4 (Uplink for PCIe Gen2) 0xEB (Gen2 PLL)
- MODPHY5 (Uplink for PCIe Gen3) 0x10 (Gen3 PLL)

Skip Gbe lane number information as not used for Lewisburg.

For Usb3 lane number information use proper lanes (only Los1~2 register base) at WM26 instead if FIA for determining Lane Ownership.

For Sata lane number information use proper lanes (only Los2~4 register base) at WM26 instead if FIA for determining Lane Ownership.

For sSata lane number information use proper lanes (Los2~3 register base) at WM26 instead if FIA for determining Lane Ownership.

For Pcie lane number information use proper lanes at WM26 instead if FIA for determining Lane Ownership.

For getting Hsio lanes use I/O Lanes Server = [0, 10, 26] at proper modphy PIDs (PID_MODPHY1 0xE9, PID_MODPHY2 0xA9) at WM26.

For getting Hsio lanes at WM20 use I/O Lanes Server = [0, 4, 20] at proper modphy PIDs at WM20.

31.30 Intel® Trace Hub

The Lewisburg will leverage all the programming from Skylake/Kaby Lake PCH-H.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

Initialize Trace Hub twice:

- Pre Policy: using Pcd to determine if enabled
- On Policy: using Policy to determine if enabled.

31.30.1 To 30.1 Intel® TH Initialization

7. Program FW_BAR (PCI offset 70h+ 74h) – aka FTMR. Refer to section 5.9.1 for the allocation. There is also a need to program the needed fabric decode per 2.5.2.1.
   a. Skip setting TRACE_HUB_STH_ACPIBAR_BASE MSR from Lewisburg RC code, and move it to CPU RC.

31.31 Management Component Transport Protocol (MCTP)

UEFI FW shall configure PSFs for MCTP.

Lewisburg pogramming guidance is listed below.

1. SPS specific configuration
   a. Register 0x4008 to value 0x00050101 - PSF-1-RC-OWNER-RS0, PMT (PSF#5, North Port Group, Port#1, Channel#1)
   b. Register 0x40D0 to value 0x00010000 - PSF-1-PSF-MC-AGENT-MCAST1-RS0-TGT0-MCTP1, DMI (PSF#1, North Port Group, Port#0, Channel#0)
   c. Register 0x40D4 to value 0x00060101 -PSF-1-PSF-MC-AGENT-MCAST1-RS0-TGT2-MCTP1, PMT IE (PSF#6, North Port Group, Port#1, Channel#1))
   d. Register 0x40D8 to value 0x00018000 - PSF-1-PSF-MC-AGENT-MCAST1-RS0-TGT1-MCTP1, SPA (PSF#1, South Port Group, Port#0, Channel#0)
   e. Register 0x4060 to value 0x00000007 - PSF-1-PSF-MC-CONTROL-MCAST1-RS0-MCTP1, Enable MCAST with three TARGETS
   f. Register 0x4008 to value 0x00050101 - PSF-6-RC-OWNER-RS0, PMT (PSF#5, North Port Group, Port#1, Channel#1)
   g. Register 0x4050 to value 0x00060101 -PSF-6-PSF-MC-AGENT-MCAST0-RS0-TGT0-MCTP0, PMT IE (PSF#6, North Port Group, Port#1, Channel#1))
   h. Register 0x4044 to value 0x00000003 - PSF-6-PSF-MC-CONTROL-MCAST0-RS0-MCTP0, Enable MCAST with one TARGET
2. Configure PSF1 RC Owner – PMT
3. Configure Broadcast to DMI
4. Configure Broadcast for IE
5. Configure MCTP Multicast Control
6. Configure PSF6 RC Owner – PMT
7. Configure PSF6 MCTP Multicast Control

Skip configuring MCTP on DWR flow (PchIsDwrFlow()).

31.32 Reliability, availability and serviceability (RAS)

Checks whether there is any PSF internal Fabric Errors Logged. If Yes just log this on the serial console and store in the HOB to be restored later in DXE by any driver/RAS driver provided by OEM.

31.32.1 PEI Phase flow

On pre-memory initialization phase log internal PSF errors:

1. Check all errors in all PSF for all channels in all ports for each port Group and RS
2. Loop each PSF
3. LBG A0 WA based on HSD request 4929460 we can't trust first error register read
4. If the second read value is different we must read it back third time in order to make sure value is correct
5. Error has been detected for this port
6. Set global error bits in global error structure
7. Finaly store all data into HOB so it can be used later in DXE by the RAS modules

31.33 Additional Lewisburg Devices

The MS Unit is a Multifunction device (device 17), consisting of MROM0 (Function 0), MROM 1(Function 1) and SSATA (Function 5).

31.33.1 sSATA Controller/Ports

Server SATA (or secondary) SATA is supported and is implemented as a separate function in MSUnit device. Therefore, in Lewisburg certain steps must be taken in order to properly recognize that sSATA and/or MROM1 (which is also a separate function of MSUnit) and in order to disable these. Although the sSATA is just other function of the MSUnit device, it is physically connected to different PSF. The BIOS must ensure that on particular PSF# port config space register shadowing is disabled. Detailed steps are provided below in suplement to 14.3 SATA Controller Disabling.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

For modifications in SATA programming see: 31.14 Integrated SATA Controllers

For Thermal Management see: 31.17 Thermal Management Support

For HSIO lanes see: 31.29 HSIO and FIA.

Extend existing SATA controller programming with SATA Controller Number to:

- use appropriate number device fumber and fuction
- use appropriate max number of ports

for all programing descibed in :

- all programing descibed in 14.1 SATA Controller Initialization
  - both on PEI and DXE phase
- all programing descibed in 14.3 SATA Controller Disabling

  For Skylake/Kaby Lake PCH-H:

  7. Set PCR[PSF_1] + "AGNT_T0_SHDW_PCIEN"[SATA][8] to 1. For sSATA use sSATA PSF base address 0C00h.

  Perform the same procedure if MS Unit Hide Control Register MROM0 PCI offset D4h[5] is enabled:

  - Set PCR[PSF_1] + "AGNT_T0_SHDW_PCIEN"[sSATA][8] to 1, to perform PSF T0 function disabling
- SATA test mode
  - use lanes 12-17 for sSATA (while 18-25 are used for SATA).
- SATA RSTe EFI OpRom:
  - If sSATA RST policy to load EfiRaidDriver is enabled:
    - install protocol guid to trigger Secondary Sata RSTe EFI OpRom load

Implement functions analogical to Sata:

- GetIsPchsSataPresent
- GetPchMaxsSataPortNum
- GetPchMaxsSataControllerNum
- sSataLaneExist

31.33.2 MROM 0 and MROM 1

These two manageability functions are present. The MROM0 has exact same reason, functionality and BIOS programming steps as described in Intel® Series 8 chipset family BIOS specification. Basically, this function allows certain functionality of the Manageability Engine to be loaded as OpROM during BIOS boot. The BIOS must ensure the OpROM is being dispatched properly.

Second MROM1 plays the same role as MROM0 but allows the loading/running of certain functionality of the Innovation Engine to be dispatched during BIOS flow. The

Lewisburg BIOS must ensure that MROM1 OpROM gets dispatched without problems.

Below can be considered duplicate with EDS:

The general feature are as follows:

- MROM0:
  - provides PCI configuration registers describe the expansion ROM region for the ME.
  - controls and enables various EVA functions
  - MS Unit Device Function Hide Control register provides the ability to hide SSATA and MROM1 from BIOS and the OS. MROM0 is never hidden.
  - provides control for the ADR Timer (Asynchronous DRAM Refresh Timer).
- MROM1 provides the configuration control of the IE via PCIe configuration registers

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

31.33.2.1 Pre-mem phase

31.33.2.1.1 ADR

UEFI FW needs to implement three BIOS options:

1) One option to choose value of ADR_TMR (BDF 0/17/0 Off 180h bits 31:30). Valid values:

25us - 0b
50us - 1b
100us - 10b
0us - 11b

2) One option to choose multiplier ADR_MULT (BDF 0/17/0 Off 180h bits 27:24). Valid values:

x1 - 0b
x8 - 1b
x24 - 10b
x40 - 11b
x56 - 100b
x64 - 101b
x72 - 110b
x80 - 111b
x88 - 1000b
x96 - 1001b
Other values are RESERVED.

3) One option to choose GPIO_B or GPIO_C (mutually exclusive)

When choosing GPIO_B:

1. Configure PM_SYNC in serial mode 0/31/2 A8h bit 9 = 1
2. Program PWRMBASE +D4h [30:29] = 1b, for GPIO_B to PM_SYNC Enable
3. Program PWRMBASE +F4h [30:29] = 1b, for GPIO_B to PM_SYNC Enable in C0
4. Program PWRMBASE +C8h [10:9] = 1b, to map CPU_GP_0 to GPIO_B
5. Program PWRMBASE +F0h [31:30] to 01b to select GPIO_B as ADR GPIO (Disable timer if requested by SV)
6. Program CPU_GP_0 as GPIO
7. Program PWRMBASE +F0h [0] = 1b ADR feature enable

When choosing GPIO_C:

1. Configure PM_SYNC in serial mode 0/31/2 A8h bit 9 = 1
2. Program PWRMBASE +D4h [30:29] = 2b, for GPIO_C to PM_SYNC Enable
3. Program PWRMBASE +F4h [30:29] = 2b, for GPIO_C to PM_SYNC Enable in C0
4. Program PWRMBASE +C8h [10:9] = 0b, to map CPU_GP_0 to GPIO_C
5. Program PWRMBASE +F0h [31:30] to 10b to select GPIO_B as ADR GPIO (Disable timer if requested by SV)
6. Program CPU_GP_0 as GPIO
7. Program PWRMBASE +F0h [0] = 1b ADR feature enable

31.33.2.1.2 MROM1

Disable MROM1 if MS Unit Hide Control Register MROM0 PCI offset D4h[1] is enabled:

- Set PCR[PSF_3] + "AGNT_T0_SHDW_PCIEN"[MROM1][8] to 1, to perform PSF T0 function disabling

31.33.2.2 End of DXE

31.33.2.2.1 MROM0

Apply MROM0 MSDEVFUNCHIDE lock while processing all locks

31.33.2.3 To 5.14 Additional Programming Steps

After completing: 5.14.1 Processor Set Strap Message:

1. Set MROM0 PCI offset D4h[31] = 1b

31.33.3 Acceleration Complex.

The AC is a set of devices that allow the system to accelerate Network Processing and speed up many encryption algorithms on the fly by HW. Please refer to EDS for all the details.

From Lewisburg BIOS these set of devices are visible as just another set of standard

PCIe 3.0 devices. Those devices like Intel® QuickAssist Technology and two Dual 10 GB Ethernet controllers have some newer PCIe enumeration requirements – like support for SR-IOV, but considering latest EDKII and UEFI BIOS core all these are supported.

31.33.3.1 Intel® QuickAssist Technology

Intel® QuickAssist Technology units support virtual functions with Bars other than 4k. BIOS will need to support non-standard SRIOV Bar Page Sizes in order to make use of this feature.

31.33.4 Innovation Engine

This Engine is another Minute IA core that allows running separate IBV firmware.

The Lewisburg BIOS must include this firmware in the separate SPI flash region with additional soft-straps configuration settings in Flash descriptor.

Second requirement for Lewisburg BIOS is to make sure the BIOS will enumerate and dispatch OpROM (and/or UEFI driver) for that device.

Possible additional detailed programming steps, if any, are TBD.

Lewisburg pogramming guidance and matching Skylake/Kaby Lake PCH-H sections are listed below.

For SVID /SID related programming see 31.2.2SVID and SID of PCH Devices.

For MCTP related programming see 31.31 Management Component Transport Protocol (MCTP).

For Demoted Warm Reset related programming see 31.4.1 Demoted Warm Reset.

§ §