



Intel® Communications Chipset 8900 to 8920 Series Software

Programmer's Guide

Rev. 006

October 2016



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

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

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

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

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

Any software source code reprinted in this document is furnished for informational purposes only and may only be used or copied and no license, express or implied, by estoppel or otherwise, to any of the reprinted source code is granted by this document.

Basis, Basis Peak, BlueMoon, BunnyPeople, Celeron, Centrino, Cilk, Curie, Flexpipe, Intel, the Intel logo, the Intel Anti-Theft technology logo, Intel AppUp, the Intel AppUp logo, Intel Atom, Intel CoFluent, Intel Core, Intel Inside, the Intel Inside logo, Intel Insider, Intel RealSense, Intel SingleDriver, Intel SpeedStep, Intel vPro, Intel Xeon Phi, Intel XScale, InTru, the InTru logo, the InTru Inside logo, InTru soundmark, Iris, Itanium, Kno, Look Inside., the Look Inside. logo, Mashery, MCS, MMX, Pentium, picoArray, Picochip, picoXcell, Puma, Quark, SMARTi, smartSignaling, Sound Mark, Stay With It, the Engineering Stay With It logo, The Creators Project, The Journey Inside, Thunderbolt, the Thunderbolt logo, Transcende, Ultrabook, VTune, Xeon, X-GOLD, XMM, X-PMU and XPOSYS are trademarks of Intel Corporation in the U.S. and/or other countries.

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

Copyright © 2010–2016, Intel Corporation. All rights reserved.



Revision History

Date	Revision	Description
October 2016	006	Update includes: <ul style="list-style-type: none"> Updated Section 1.3 Product Documentation
March 2016	005	Updates include: <ul style="list-style-type: none"> Updated Stateful Compression Level Details on page 58 and Stateless Compression Level Details on page 58 Added DRBG_POLL_AND_WAIT optional build flag to Build Flag Summary on page 61
October 2015	004	Updates include: <ul style="list-style-type: none"> Updated Build Flag Summary on page 61 Updated Cryptographic Logical Instance Parameters on page 75 and Data Compression Logical Instance Parameters on page 76 Updated Intel® QuickAssist Technology API Limitations on page 98
March 2015	003	Updates include: <ul style="list-style-type: none"> Added Intel® QuickAssist Technology Entries in the /proc Filesystem on page 42 Added How to Call the Heartbeat Query on page 47 Updated Build Flag Summary on page 61 Added Acceleration Driver Return Codes on page 66 Updated Dynamic Instance Configuration Example on page 77 Updated Maximum Number of Process Calculations on page 79 and Resubmitting After Getting an Overflow Error on page 100
December 2014	002	Updates include: <ul style="list-style-type: none"> Added Intel® QuickAssist Technology Compression API Errors on page 53 Updated Intel® QuickAssist Technology API Limitations on page 98 Added Resubmitting After Getting an Overflow Error on page 100 Added new APIs to Dynamic Instance Allocation Functions on page 107 Updated Reset Device Function on page 134 Added Thread-less APIs on page 135 Other general updates.
July 2014	001	Updates include: <ul style="list-style-type: none"> First "public" version of the document. Based on "Intel Confidential" document number 441782-1.8 with the revision history of that document retained for reference purposes.
May 2014	1.8	Updates include: <ul style="list-style-type: none"> Added Compiling with Debug Symbols on page 66
March 2014	1.7	Updates include: <ul style="list-style-type: none"> Added new information to "direct user space access" bullet in Acceleration Drivers Overview on page 28 Added further detail to note in Hardware Assisted Rings on page 28 Updated Linux* Software Context for Acceleration Drivers on page 30 Added Stateless Compression Level Details on page 58 Added support for the PF/VF concurrency for SRIOV_Enabled in General Parameters on page 70
continued...		



Date	Revision	Description
		<ul style="list-style-type: none"> Added Dynamic Compression for Data Compression Service on page 101, Maximal Expansion with Auto Select Best Feature for Data Compression Service on page 102, and Maximal Expansion and Destination Buffer Size
December 2013	1.6	Updates include: <ul style="list-style-type: none"> Added new information to Intel® QuickAssist Technology API Limitations on page 98 Added Running Applications as Non-Root User on page 63 Added Compiling Acceleration Software on Older Kernels on page 65 Changed document and software title to specify chipset SKU range. Other minor updates.
August 2013	1.5	Updates include: <ul style="list-style-type: none"> Added section, "Heartbeat Feature in a Virtualized Environment" Removed two stateful compression/decompression limitations from Intel® QuickAssist Technology API Limitations on page 98 Added new NRBG and DRBG support information to Random Number Generation Functions on page 122
June 2013	1.4	Updates for software release 1.3.0: <ul style="list-style-type: none"> Added Support for Multiple Acceleration Hardware Generations on page 25 Added Compression Status Codes on page 53 Updated Stateful Compression - Dealing with Error Code CPA_DC_BAD_LITLEN_CODES (-7) on page 54 and subsections Added Stateful Compression Level Details on page 58 Updated Build Flag Summary on page 61 to add ICP_TRACE option Updated icp_sal_CyPollInstance on page 119 Updated icp_sal_DcPollInstance on page 120
March 2013	1.3	Updates for software release 1.2.0: <ul style="list-style-type: none"> In General Parameters, added SRIOV_Enable and PF_bundle_offset Added [DYN] Section Updated Sample Configuration File (V2) Added Driver Threading Model Added Stateful Compression - Dealing with Error Code CPA_DC_BAD_LITLEN_CODES (-7) Added Acceleration Driver Error Scenarios Added Build Flag Summary Added Dynamic Instance Allocation Functions Added IOMMU Remapping Functions
December 2012	1.2	Updates for software release 1.1.0: <ul style="list-style-type: none"> Updated Heartbeat Feature and Recovery from Hardware Errors Added User Proc Entry Read (not Enabled by Default) Added User Application Heartbeat APIs (not Enabled by Default) Updated Intel QuickAssist Technology API Limitations to better clarify autoSelectBest behavior for static compression service Added GbE Watchdog Service Added Special Considerations When Using the Heartbeat Feature and GbE Added icp_sal_drbgGetInstance Updated DRBG Health Test and cpaCyDrbgSessionInit Implementation Detail Added User Space Heartbeat Functions
October 2012	1.1	Updates for software release 1.0.1: <ul style="list-style-type: none"> Added Heartbeat Feature and Recovery from Hardware Errors Updated General Parameters
continued...		



Date	Revision	Description
		<ul style="list-style-type: none"> Updated Cryptographic Logical Instance Parameters Updated Data Compression Logical Instance Parameters Added DRBG HealthTest and cpaCyDrbgSessionInit Implementation Detail
September 2012	1.0	Corresponds with software release 1.0.0



Contents

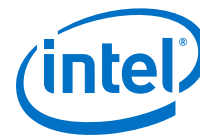
Revision History	3
Part 1: Overview	13
1.0 Introduction	14
1.1 Terminology.....	14
1.2 Document Organization.....	14
1.3 Product Documentation.....	14
1.4 Typographical Conventions.....	15
2.0 Platform Overview	16
2.1 Platform Synopsis.....	16
2.2 Determining the PCH SKU Type.....	18
2.3 Determining the PCH Device Stepping.....	19
3.0 Software Overview	21
3.1 High-Level Software Architecture Overview.....	21
3.2 Logical Instances.....	23
3.2.1 Response Processing.....	23
3.2.1.1 Interrupt Mode.....	23
3.2.1.2 Polled Mode.....	24
3.3 Operating System Support.....	25
3.4 OpenSSL* Library Inclusion and Usage.....	25
3.5 Support for Multiple Acceleration Hardware Generations.....	25
Part 2: Acceleration Drivers	27
4.0 Acceleration Drivers Overview	28
4.1 Hardware Assisted Rings.....	28
4.2 Basic Software Context for Acceleration Drivers.....	30
4.3 Linux* Software Context for Acceleration Drivers.....	30
4.4 Acceleration Drivers.....	31
4.4.1 Framework Overview.....	32
4.4.2 Service Access Layer.....	32
4.4.3 Acceleration Driver Framework.....	33
4.4.4 Acceleration Driver Configuration File.....	33
4.4.5 Utility for Loading Configuration Files and Sending Events to the Driver - adf_ctl.....	34
4.5 Acceleration Architecture in Kernel and User Space.....	35
4.5.1 User Space Memory Allocation.....	36
4.5.1.1 Accelerator Driver Memory Allocation.....	36
4.5.1.2 Application Payload Memory Allocation.....	37
4.5.2 User Space Additional Functions.....	38
4.5.3 User Space Configuration.....	39
4.5.4 User Space Response Processing.....	40
4.5.4.1 User Space Interrupt Mode.....	40
4.5.4.2 User Space Polled Mode.....	41



4.6 Managing Acceleration Devices Using qat_service.....	41
4.7 Intel® QuickAssist Technology Entries in the /proc Filesystem.....	42
4.8 Debug Feature.....	44
4.9 Heartbeat Feature and Recovery from Hardware Errors.....	46
4.9.1 How to Call the Heartbeat Query.....	47
4.9.1.1 User Proc Entry Read (not Enabled by Default).....	47
4.9.1.2 User Application Heartbeat APIs (not Enabled by Default).....	48
4.9.2 Handling Heartbeat Failures.....	49
4.9.3 AER and Uncorrectable Errors.....	49
4.9.4 Handling Device Failures in a Virtualized Environment.....	50
4.9.5 GbE Watchdog Service.....	51
4.9.6 Special Considerations When Using the Heartbeat Feature and the GbE Watchdog Service.....	51
4.10 Driver Threading Model.....	52
4.10.1 Thread-less Mode.....	53
4.11 Compression Status Codes.....	53
4.11.1 Intel® QuickAssist Technology Compression API Errors.....	53
4.12 Stateful Compression - Dealing with Error Code CPA_DC_BAD_LITLEN_CODES (-7)	54
4.12.1 Example of a Stream that Triggers Error Code (-7).....	55
4.12.2 Special Case when a Packet Cuts a Header in the Stream.....	56
4.12.3 Pseudo Code for Handling Error Code -7.....	56
4.12.4 Unprocessed Data During Stateful Decompression Operations.....	57
4.13 Stateful Compression Level Details.....	58
4.14 Stateless Compression Level Details.....	58
4.15 Acceleration Driver Error Scenarios.....	59
4.15.1 User Space Process Crash.....	59
4.15.2 Hardware Hang Detected by Heartbeat.....	59
4.15.3 Hardware Error Detected by AER.....	60
4.15.4 Virtualization: User Space Process Crash (in Guest OS).....	60
4.15.5 Virtualization: Guest OS Kernel Crash.....	60
4.15.6 Virtualization: Hardware Hang Detected by Heartbeat.....	61
4.15.7 Virtualization: Hardware Hang Detected by AER.....	61
4.16 Build Flag Summary.....	61
4.17 Running Applications as Non-Root User.....	63
4.18 Compiling Acceleration Software on Older Kernels.....	65
4.19 Compiling with Debug Symbols.....	66
4.20 Acceleration Driver Return Codes.....	66
5.0 Acceleration Driver Configuration File.....	69
5.1 Configuration File Overview.....	69
5.2 General Section.....	70
5.2.1 General Parameters.....	70
5.2.2 Statistics Parameters.....	73
5.2.3 Optimized Firmware for Wireless Applications.....	74
5.3 Logical Instances Section.....	74
5.3.1 [KERNEL] Section.....	75
5.3.1.1 Cryptographic Logical Instance Parameters.....	75
5.3.1.2 Data Compression Logical Instance Parameters.....	76
5.3.2 [DYN] Section.....	77
5.3.2.1 Dynamic Instance Configuration Example.....	77
5.3.3 User Process [xxxxx] Sections.....	78



5.3.3.1 Maximum Number of Process Calculations.....	79
5.4 Configuring Multiple PCH Devices in a System.....	80
5.5 Configuring Multiple Processes on a Multiple-Device System.....	81
5.6 Sample Configuration File (V2).....	84
5.7 Configuration File Version 2 Differences.....	89
6.0 Secure Architecture Considerations.....	90
6.1 Terminology.....	90
6.1.1 Threat Categories.....	90
6.1.2 Attack Mechanism.....	90
6.1.3 Attacker Privilege.....	91
6.1.4 Deployment Models.....	91
6.2 Threat/Attack Vectors.....	92
6.2.1 General Mitigation.....	92
6.2.2 General Threats.....	92
6.2.2.1 DMA.....	93
6.2.2.2 Intentional Modification of IA Driver.....	93
6.2.2.3 Modification of Intel® QuickAssist Accelerator Firmware.....	94
6.2.2.4 Modification of the PCH Configuration File.....	94
6.2.2.5 Malicious Application Code.....	94
6.2.2.6 Contrived Packet Stream.....	94
6.2.3 Threats Against the Cryptographic Service.....	95
6.2.3.1 Reading and Writing of Cryptographic Keys.....	95
6.2.3.2 Modification of Public Key Firmware.....	95
6.2.3.3 Failure of the Entropy Source for the Random Number Generator.....	96
6.2.3.4 Interference Among Users of the Random Number Service.....	96
6.2.4 Data Compression Service Threats.....	96
6.2.4.1 Read/Write of Save/Restore Context.....	96
6.2.4.2 Stateful Behavior.....	96
6.2.4.3 Incomplete or Malformed Huffman Tree.....	97
6.2.4.4 Contrived Packet Stream.....	97
7.0 Supported APIs.....	98
7.1 Intel® QuickAssist Technology APIs.....	98
7.1.1 Intel® QuickAssist Technology API Limitations.....	98
7.1.1.1 Resubmitting After Getting an Overflow Error.....	100
7.1.1.2 Dynamic Compression for Data Compression Service	101
7.1.1.3 Maximal Expansion with Auto Select Best Feature for Data Compression Service	102
7.1.1.4 Maximal Expansion and Destination Buffer Size	103
7.1.2 Data Plane APIs Overview.....	103
7.1.2.1 IA Cycle Count Reduction When Using Data Plane APIs.....	104
7.1.2.2 Usage Constraints on the Data Plane APIs.....	105
7.1.2.3 Cryptographic and Data Compression API Descriptions.....	106
7.2 Additional APIs.....	106
7.2.1 Dynamic Instance Allocation Functions.....	107
7.2.1.1 icp_sal_userCyGetAvailableNumDynInstances.....	108
7.2.1.2 icp_sal_userDcGetAvailableNumDynInstances.....	108
7.2.1.3 icp_sal_userCyInstancesAlloc.....	109
7.2.1.4 icp_sal_userDcInstancesAlloc.....	109
7.2.1.5 icp_sal_userCyFreeInstances.....	110
7.2.1.6 icp_sal_userDcFreeInstances.....	110



7.2.1.7 icp_sal_userCyGetAvailableNumDynInstancesByDevPkg.....	111
7.2.1.8 icp_sal_userDcGetAvailableNumDynInstancesByDevPkg.....	112
7.2.1.9 icp_sal_userCyInstancesAllocByDevPkg.....	112
7.2.1.10 icp_sal_userDcInstancesAllocByDevPkg.....	113
7.2.1.11 icp_sal_userCyGetAvailableNumDynInstancesByPkgAccel.....	114
7.2.1.12 icp_sal_userCyInstancesAllocByPkgAccel.....	114
7.2.2 IOMMU Remapping Functions.....	115
7.2.2.1 icp_sal_iommu_get_remap_size.....	115
7.2.2.2 icp_sal_iommu_map.....	116
7.2.2.3 icp_sal_iommu_unmap.....	116
7.2.2.4 IOMMU Remapping Function Usage.....	117
7.2.3 Polling Functions.....	117
7.2.3.1 icp_sal_pollBank.....	118
7.2.3.2 icp_sal_pollAllBanks.....	119
7.2.3.3 icp_sal_CyPollInstance.....	119
7.2.3.4 icp_sal_DcPollInstance.....	120
7.2.3.5 icp_sal_CyPollDpInstance.....	121
7.2.3.6 icp_sal_DcPollDpInstance.....	121
7.2.4 Random Number Generation Functions.....	122
7.2.4.1 icp_sal_drbgGetEnropyInputFuncRegister.....	123
7.2.4.2 icp_sal_drbgGetInstance.....	124
7.2.4.3 icp_sal_drbgGetNonceFuncRegister.....	124
7.2.4.4 icp_sal_drbgHTGenerate.....	125
7.2.4.5 icp_sal_drbgHTGetTestSessionSize.....	125
7.2.4.6 icp_sal_drbgHTInstantiate.....	126
7.2.4.7 icp_sal_drbgHTReseed.....	126
7.2.4.8 icp_sal_drbgIsDFReqFuncRegister.....	127
7.2.4.9 icp_sal_nrbgHealthTest.....	127
7.2.4.10 DRBG Health Test and cpaCyDrbgSessionInit Implementation Detail.....	128
7.2.5 User Space Access Configuration Functions.....	129
7.2.5.1 icp_sal_userStart.....	129
7.2.5.2 icp_sal_userStartMultiProcess.....	130
7.2.5.3 icp_sal_userStop.....	131
7.2.6 User Space Heartbeat Functions.....	132
7.2.6.1 icp_sal_check_device.....	132
7.2.6.2 icp_sal_check_all_devices.....	133
7.2.7 Version Information Function.....	133
7.2.7.1 icp_sal_getDevVersionInfo.....	133
7.2.8 Reset Device Function.....	134
7.2.8.1 icp_sal_reset_device.....	134
7.2.9 Thread-less APIs.....	135
7.2.9.1 icp_sal_poll_device_events.....	135
7.2.9.2 icp_sal_find_new_devices.....	135

Part 3: Applications and Usage Models..... 137

8.0 Application Usage Guidelines..... 138

8.1 Mapping Service Instances to Hardware Accelerators on the PCH.....	138
8.1.1 Processor and PCH Device Communication.....	139
8.1.2 Service Instances and Interaction with the Hardware.....	140



8.1.3 Service Instance Configuration.....	141
8.1.4 Guidelines for Using Multiple Intel® QuickAssist Instances for Load Balancing in Cryptography Applications.....	142
8.2 Cryptography Applications.....	145
8.2.1 IPsec and SSL VPNs.....	145
8.2.2 Encrypted Storage.....	146
8.2.3 Web Proxy Appliances.....	146
8.3 Data Compression Applications.....	147
8.3.1 Compression for Storage.....	147
8.3.2 Data Deduplication and WAN Acceleration.....	147
Appendix A Acceleration Driver Configuration File - Earlier File Format.....	149
A.1 Configuration File Overview.....	149
A.2 General Section.....	150
A.2.1 General Parameters.....	151
A.2.2 QAT Parameters.....	151
A.2.3 Statistics Parameters.....	152
A.3 [AcceleratorX] Section.....	153
A.3.1 Interrupt Coalescing Parameters.....	153
A.3.2 Affinity Parameters.....	154
A.4 Logical Instances Section.....	155
A.4.1 [KERNEL] Section.....	155
A.4.1.1 Cryptographic Logical Instance Parameters.....	156
A.4.1.2 Data Compression Logical Instance Parameters.....	157
A.4.2 User Process Instance [xxxxx] Sections.....	158
A.5 Sample Configuration File (V1).....	159
Appendix B Glossary.....	167



Figures

1	I/O Optimized Platform Example.....	17
2	Bladed Platform Example.....	18
3	PCH SKU Identification Example.....	19
4	Software Architecture Overview.....	21
5	Kernel Space Response Ring Processing.....	24
6	Intel® QuickAssist Accelerator Ring Access.....	29
7	Ring Partitioning on the Chipset Device.....	29
8	Basic Software Context.....	30
9	Linux Software Context.....	31
10	Acceleration Driver Framework.....	32
11	Software Architecture for Kernel and User Space.....	36
12	User Space Memory Allocation at Initialization.....	37
13	User Space Process with Two Logical Instances.....	39
14	User Space Response Processing for Interrupt Mode.....	41
15	Stream of Compressed Data Split into Three Packets	55
16	Accelerator Software Consumes Data up to End of Header 2 where Error Code (-7) is Normally Generated	55
17	Unprocessed Data Appended to Next Packet	56
18	Packet Cut in the Middle of a Header.....	56
19	New Format of Packet 3 with Data Prepended from Previous Requests.....	56
20	Unprocessed Data for Incomplete Header.....	58
21	Ring Banks.....	69
22	Dynamic Compression Data Path.....	101
23	Amortizing the Cost of an MMIO Across Multiple Requests.....	105
24	Processor and PCH Device Components.....	139
25	Processor and PCH Device Communication.....	140
26	Service Instance Attributes and Hardware Components.....	141
27	Service Instance Configuration.....	142
28	Entities and Relationships for Load Balancing.....	143
29	Load Balancing Scenarios.....	144
30	Ring Banks.....	150
31	Ring Bank Affinity to Core for MSI-X Interrupts.....	154



Tables

1	Device Enumeration Example.....	34
2	Heartbeat/GbE Watchdog Service Scenarios.....	51
3	Intel® QuickAssist Technology Compression API Errors.....	53
4	Required Build Flags.....	61
5	Optional Build Flags.....	62
6	General Parameters.....	70
7	Statistics Parameters.....	73
8	Cryptographic Logical Instance Parameters.....	76
9	User Process [xxxxx] Sections Parameters.....	79
10	System Threat Categories.....	90
11	Attack Mechanisms and Examples.....	91
12	Attacker Privilege.....	91
13	Deployment Models.....	92
14	Compression/Decompression Overflow Behavior	100
15	Service Instance Attributes.....	141
16	General Parameters - Earlier File Format.....	151
17	QAT Parameters - Earlier File Format.....	152
18	Statistics Parameters.....	152
19	Interrupt Coalescing Parameters - Earlier File Format.....	153
20	Ring Bank Affinity Parameters.....	155
21	Cryptographic Logical Instance Parameters - Earlier File Format.....	156



Part 1: Overview



1.0 Introduction

This Programmer's Guide provides information on the architecture of the software and usage guidelines. Information on the use of Intel® QuickAssist Technology APIs, which provide the interface to acceleration services (cryptographic, data compression), is documented in the related QuickAssist Technology Software Library documentation (see [Product Documentation](#)).

1.1 Terminology

In this document, for convenience:

- Software package is used as a generic term for the Intel® Communications Chipset 8900 to 8920 Series software package.
- Platform Controller Hub (PCH) is used as a generic term for the Intel® Communications Chipset 8900 to 8920 Series.
- Accelerator is used as a generic term for the Intel® QuickAssist Accelerator device(s) integrated in the Intel® Communications Chipset 8900 to 8920 Series.
- Acceleration drivers is used as a generic term for the software that allows the QuickAssist Software Library APIs to access the Intel® QuickAssist Accelerator device(s) integrated in the Intel® Communications Chipset 8900 to 8920 Series.
- Mobile platform is used as a generic term for a platform that combines Intel® Xeon® and Intel® Core™ Processors For Communications Infrastructure with the Intel® Communications Chipset 8900 to 8920 Series.
- Server platform is used as a generic term for a platform that combines Intel® Xeon® Processors with the Intel® Communications Chipset 8900 to 8920 Series.

Refer to [Glossary](#) on page 167 for the definition of acronyms and other terms used in this document.

1.2 Document Organization

This document is organized as follows:

- Part 1: Provides an overview of the supported hardware and an overview of the software architecture.
- Part 2: Describes the core and chipset drivers provided in the software package.
- Part 3: Describes the acceleration drivers included in the software package.
- Part 4: Provides information on specific applications and software usage models.

A glossary of the terms and acronyms used in this guide is provided at the end of the document.

1.3 Product Documentation

Documentation supporting the software package includes:



- *Intel® Communications Chipset 8900 to 8920 Series Software Release Notes*
- *Intel® Communications Chipset 8900 to 8920 Series Software for Linux* Getting Started Guide*
- *Intel® Communications Chipset 89xx Series Software for FreeBSD* Getting Started Guide*
- *Intel® Communications Chipset 89xx Series Software for Windows* Getting Started Guide*
- *Intel® Communications Chipset 8900 to 8920 Series Software Programmer's Guide (this document)*

Related QuickAssist Technology Software Library documentation includes:

- *Intel® QuickAssist Technology API Programmer's Guide*
- *Intel® QuickAssist Technology Cryptographic API Reference Manual*
- *Intel® QuickAssist Technology Data Compression API Reference Manual*

Other related documentation:

- *Intel® QuickAssist Technology Acceleration Software OS Porting Guide*
- *Using Intel® Virtualization Technology (Intel® VT) with Intel® QuickAssist Technology Application Note*
- *Intel® Communications Chipset 89xx Series External Design Specification (EDS)*
- *Supported Ethernet PHY Devices for the Intel® Communications Chipset 89xx Series Application Note*
- [Intel® 82580 Quad/Dual Gigabit Ethernet Controller Data Sheet](#)
- *Intel® Xeon® Processor (storage) - External Design Specification (EDS) Addendum - Rev. 1.1 (Reference: 503997)*

1.4 Typographical Conventions

The following conventions are used in this manual:

- `Courier font` - file names, path names, code examples, command line entries, API names, parameter names and other programming constructs
- *Italic text* - key terms and publication titles
- **Bold text** - graphical user interface entries and buttons



2.0 Platform Overview

The mobile and server platforms described in this manual are follow ons to previous generation platforms that continue to reduce power, reduce footprint and increase performance for communications infrastructure systems. The platforms deliver leadership solutions with GB/s Ethernet* MACs and Intel® QuickAssist Technology hardware: the acceleration for cryptography and data compression.

2.1 Platform Synopsis

At a high level, the platform pairs an Intel® architecture processor with the Intel® Communications Chipset 8900 to 8920 Series. Functionally, Intel® Communications Chipset 8900 to 8920 Series can be most easily described as a Platform Controller Hub (PCH) that includes both standard PC interfaces (for example, PCI Express*, SATA, USB and so on) together with accelerator and I/O interfaces (for example, Intel® QuickAssist Accelerator and GigE).

- For I/O-optimized applications, Intel® Xeon® and Intel® Core™ Processors For Communications Infrastructure are paired with the Intel® Communications Chipset 8900 to 8920 Series. [Figure 1](#) on page 17 is a block diagram of the Intel® Xeon® Processor E3-1125C with Intel® Communications Chipset 8910 Development Kit, codenamed Stargo. See the *Intel® Xeon® Processor E3-1125C with Intel® Communications Chipset 8910 Development Kit User Guide* for detailed information.
- For bladed applications, Intel® Xeon® Processors are paired with the Intel® Communications Chipset 8900 to 8920 Series. [Figure 2](#) on page 18 is a block diagram of the Intel® Xeon® Processor E5-2658 and E5-2448L with Intel® Communications Chipset 89xx Development Kit, codenamed Shumway. See the *Intel® Xeon® Processor E5-2658 and E5-2448L with Intel® Communications Chipset 8920 Development Kit User Guide* for detailed information.

Figure 1. I/O Optimized Platform Example

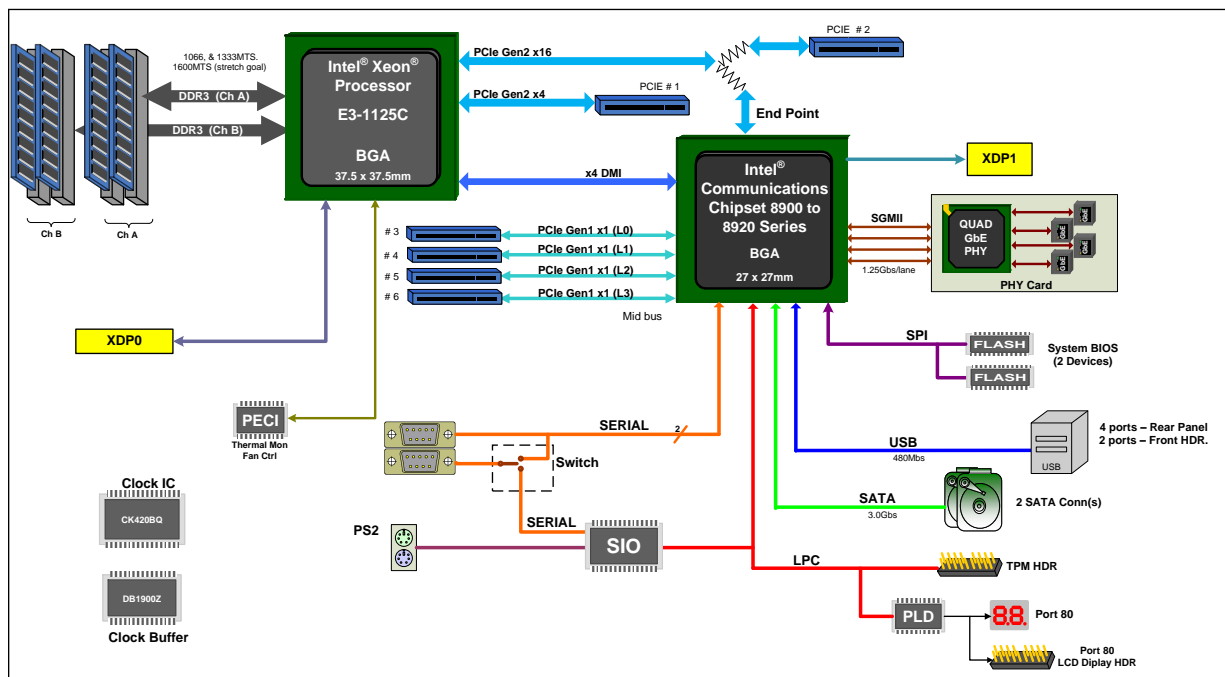
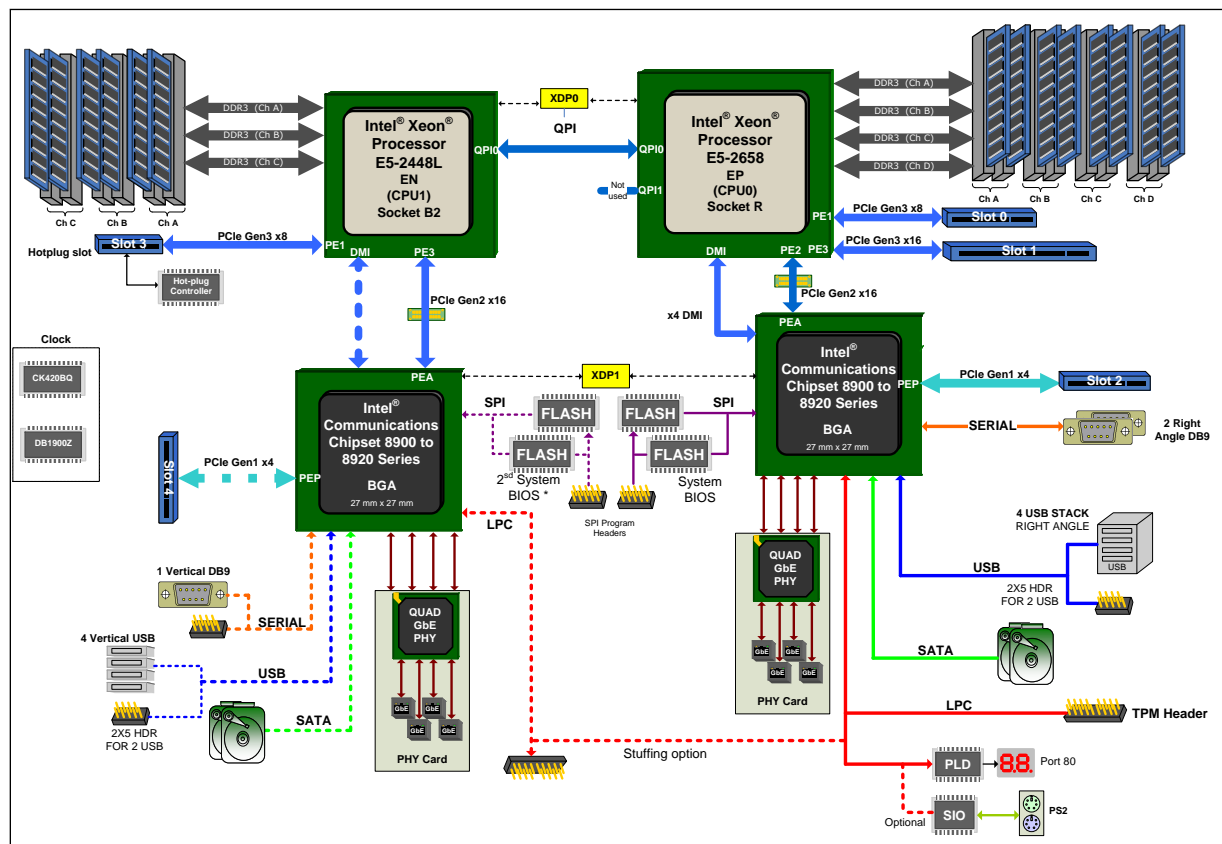


Figure 2. Bladed Platform Example



2.2 Determining the PCH SKU Type

Determine the PCH SKU type as follows:

1. Find out the bus, slot and function of the PCH devices:

```
[root@localhost ~]# lspci -d 8086:0434
```

```
03:00.0 Co-processor: Intel Corporation Device 0434 (rev 10)
82:00.0 Co-processor: Intel Corporation Device 0434 (rev 10)
```

This displays the PCI configuration space for the 0434 device. In the case of the first entry, the bus number=0x03, the device number=0x0 and the function number=0x0.

2. Read the config space using the command:

```
[root@localhost ~]# od -tx4 -Ax /proc/bus/pci/03/00.0
```

where:

- -tx4 displays the output in a readable 4-bytes word format



- `-Ax` specifies Hex. format
3. Read the `0x00040` offset specifically using the command:

```
[root@localhost ~]# od -tx4 -Ax /proc/bus/pci/03/00.0 | grep 000040
```

This gives an output similar to the following:

```
000040 00000000 00000000 00010000 0bb80000
```

Note: The word starts at 0x4C.

4. Read the element returned from the following command:

```
# od -tx4 -Ax /proc/bus/pci/03/00.0 | grep "^000040" | awk '{print $2}'
```

This gives an output similar to the following:

0bb80000

Example

Specific bits in this output determine the SKU type depending on the silicon stepping as indicated in the following table.

Silicon	Bits to Check	SKU Type
A0	17:16 = 00	SKU 4
B0	22:19 = 0111	SKU 4
	22:19 = 0110 and 17:16 = 01	SKU 3
	22:19 = 0110 and 17:16 = 10	SKU 2
Cx	Same as B0.	

Assuming a B0 stepping device, if the 0x0bb80000 output from the command is analyzed in binary form as shown in the following figure, it can be determined that bits 22:19 are 0111, indicating SKU 4.

Figure 3. PCH SKU Identification Example

0				b				b				8				0				0				0				0							
Bit 32				Bit 28				Bit 24				Bit 20				Bit 16				Bit 12				Bit 8				Bit 4				Bit 0			
0	0	0	0	0	1	0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				

2.3 Determining the PCH Device Stepping

Determine the PCH stepping as follows:



1. Find out the bus, device, and function of the PCH device.
2. Read the config space using the command:

```
# od -tx1 -Ax /proc/bus/pci/<bus number>/<device number>.<function number>
```

3. Look at offset 0x08 (Revision ID register for the device) from the beginning of PCI Configuration Space for the PCH device.

The following is the bit definition of the *Revision ID* register, an 8-bit register with bits[07:00].

bits[07:04] identify the "Major Revision":

```
0000 = A stepping
0001 = B stepping
0010 = C stepping
0011 = D stepping
```

bits[03:00] identify the "Minor Revision":

```
0000 = x0 stepping
0001 = x1 stepping
0010 = x2 stepping
0011 = x3 stepping
```

Example

For example, if you find the PCH device at bus number 02, device number 00 and function 0 then, the command to enter is:

```
# od -tx1 -Ax /proc/bus/pci/02/00.0 | grep 000000
```

This gives an output similar to the following:

```
000000 86 80 34 04 06 00 10 00 00 00 00 40 0b 00 00 80 00
```

[0x08] = 0x00, which is 0000_0000, in binary form bits[07:00]:

- bits[07:04] is the Major Revision, 0000 indicates an A stepping.
- bits[03:00] is the Minor Revision, 0000 indicates an x0 stepping.

Therefore, the PCH device is an A0 stepping.



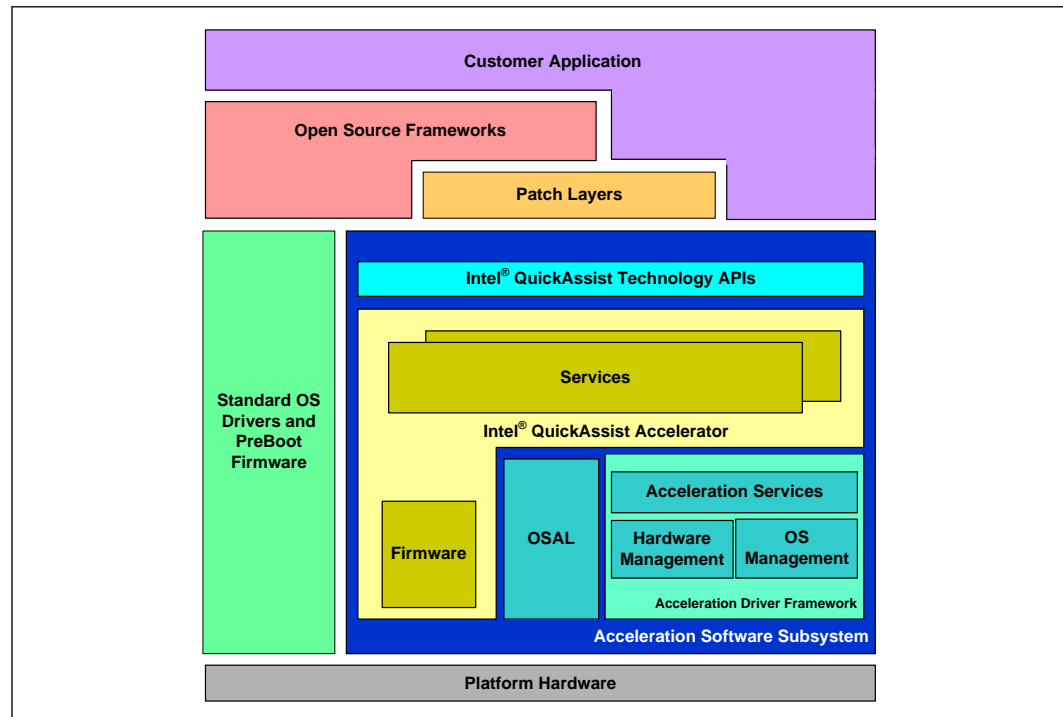
3.0 Software Overview

In addition to the hardware mentioned in [Platform Overview](#), the respective platforms have critical software components that are part of the offering. The software includes drivers and acceleration code that runs on the Intel® architecture (IA) CPUs and on the accelerators in the PCH.

3.1 High-Level Software Architecture Overview

The primary components that describe the high-level architecture are shown in the following figure.

Figure 4. Software Architecture Overview



The main software components are:

- **Pre-boot Firmware**

The Intel® Communications Chipset 8900 to 8920 Series(PCH) pre-boot firmware (provided by an IBV) executes when the system is reset or powered up. It initializes and configures system memory, chipset functions, interrupts, console devices, disk devices, integrated I/O controllers, PCI buses and devices, and additional application processors (AP) if present. IBV pre-boot firmware solutions are available to support both the legacy BIOS interface and the newer Unified Extensible Firmware Interface (UEFI).



- **Standard OS Drivers**

These drivers (provided in a standard OS distribution) include support for standard peripherals on a traditional Intel® architecture platform such as USB, SATA, Ethernet and so on. Intel provides a patch to the OS so that it recognizes the Device IDs (DIDs).

- **Acceleration Software Subsystem**

A subsystem (provided by Intel) which includes the software components that provide acceleration to applications running on the PCH. It contains the following:

- **Services (Cryptographic, Data Compression)**

Includes the firmware that drives the various workload slices in the accelerators, and the associated Intel® architecture Service libraries that expose these workloads via APIs. The Service libraries use the Acceleration Driver Framework (ADF) to plug into the OS and gain access to the hardware to communicate with the firmware. The architecture for this subsystem is detailed in : [Acceleration Drivers](#) on page 27 of this manual.

- **Intel® QuickAssist Technology APIs**

The Intel® QuickAssist Technology APIs provide service level interfaces for customer applications or Ecosystem Middleware to access the accelerator(s) in the PCH. More detail on the APIs and associated architecture is detailed in : “Acceleration Drivers” of this manual.

- **Acceleration Driver Framework (ADF)**

The Acceleration Driver Framework (ADF) includes infrastructure libraries that provide various services to the different software components of the acceleration drivers. The software framework is used to provide the acceleration services API to the application. A configuration file enables customization of system operation. See [Configuration File Overview](#) on page 69 for more information.

- **Open Source Frameworks**

This layer includes open source stacks, such as the Linux Kernel Crypto framework, zlib, and OpenSSL. The software package works to integrate the Intel® QuickAssist Technology APIs with these stacks using patch layers. These open source stacks are not developed or provided by Intel.

- **Patch Layers**

As described above, the PCH integrates with different OS stacks and Ecosystem Middleware using patch layers (translation layers). These patch layers may be developed by Intel or ecosystem vendors.

- **Customer Applications**

Customer applications may connect to the Services directly via the Intel® QuickAssist Technology API or may connect through the supported open source frameworks and associated patches.

Such applications can migrate to the PCH with little or no change provided that the Intel® QuickAssist Technology APIs are integrated with the OS stack or middleware used.



3.2 Logical Instances

A logical instance may be thought of as a channel to the hardware. A logical instance allows an address domain (that is, kernel space and individual user space processes) to configure the rings to be used by that address domain and to define the behavior of that ring.

3.2.1 Response Processing

In the kernel space, each logical instance can be configured to operate in one of the two modes:

- Interrupt mode
- Polled mode

In the user space, each logical instance can be configured to operate in one of the two modes:

- Polled mode
- Interrupt mode

3.2.1.1 Interrupt Mode

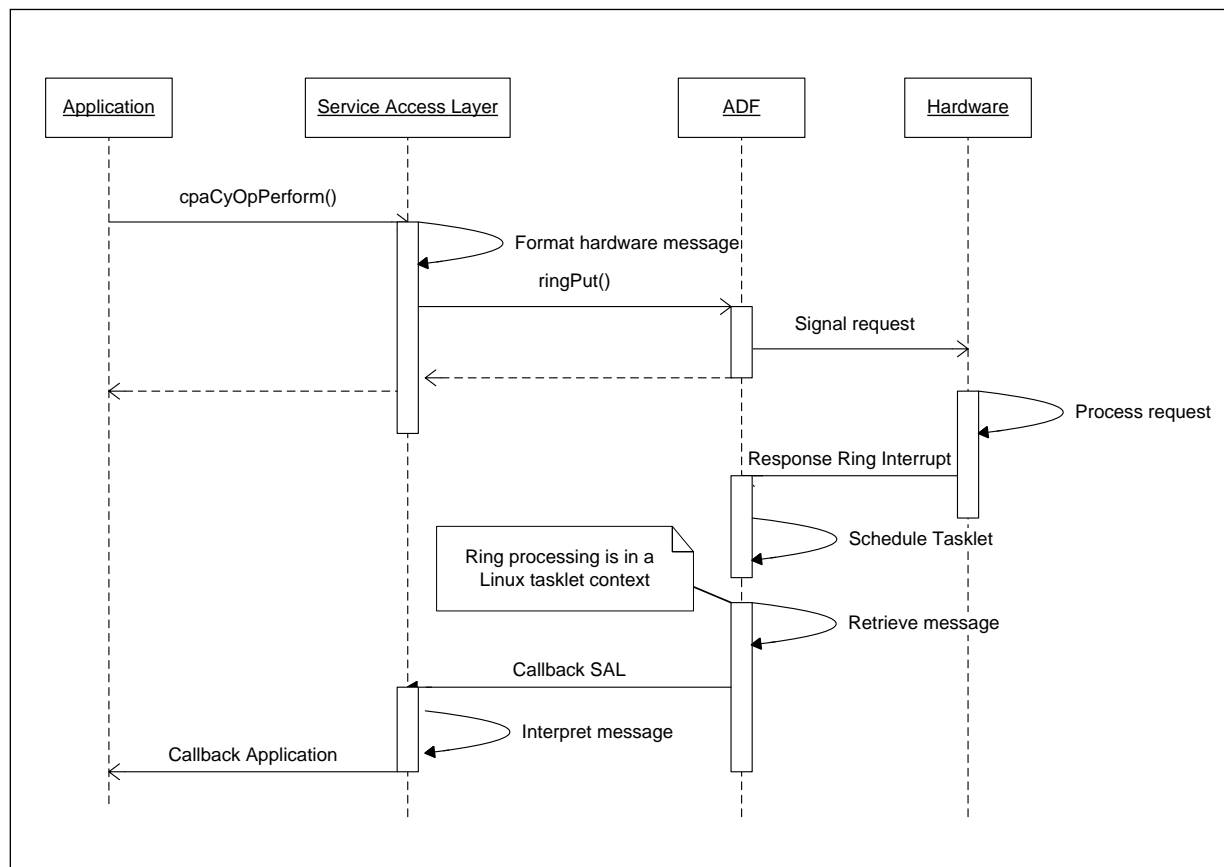
The interrupt is supported in both Kernel and User space.

When configured in interrupt mode, the Accelerator Driver Framework (ADF) registers an interrupt handler for response ring processing.

As the latency in servicing an interrupt may be costly, the hardware assisted ring provides a mechanism to amortize the cost of an interrupt into a single interrupt that may service multiple responses. The interrupt coalescing section of the configuration file allows the user to select the mechanism to amortize response interrupts using either a time-based interrupt scheme or a number-of-responses-based scheme.

The ADF registers an interrupt handler to service the ring bank interrupt. When an interrupt fires, the ADF services the interrupt and creates an interrupt handler bottom half¹ to consume the responses from the response ring. When MSI-X is supported, the bottom half of the interrupt handler is created and affinitized to the configured core. Configuration of this feature is available in the legacy variant of the configuration file only; see [Interrupt Coalescing Parameters](#) on page 153 for details. Callbacks to the application code occur in the context of this tasklet. This sequence is shown in the following figure (the full sequence has been reduced for clarity).

¹ Linux (and other operating systems) split an interrupt handler into two halves. The so-called "top half" is the routine that actually responds to the interrupt, that is, the one you register with `request_irq`. The "bottom half" is a routine that is scheduled by the top half to be executed later, at a safer time.

Figure 5. Kernel Space Response Ring Processing


3.2.1.2 Polled Mode

If the cost of servicing an interrupt and scheduling the interrupt handler bottom half is not desired, a user can choose to disable interrupts and poll for responses. This mechanism can be configured on a per logical instance basis by setting the `CyXIsPolled` or `DcXIsPolled` attribute of a logical instance in the configuration file to 1. See [Cryptographic Logical Instance Parameters](#) on page 75 and [Data Compression Logical Instance Parameters](#) on page 76 for more information. When configured to 1, the ADF does not service interrupts for that logical instance.

The ADF provides a set of APIs to allow the client to poll a single bank or all banks on a given accelerator:

- `icp_sal_pollBank` - Poll the rings on the given bank number for a given accelerator.
- `icp_sal_pollAllBanks` - Poll the rings on all banks for a given accelerator.

The Service Access Layer (SAL) provides an API to poll on an individual logical instance:

- `icp_sal_CyPollInstance` - Poll a specific cryptographic (Cy) logical instance
- `icp_sal_DcPollInstance` - Poll a specific data compression (Dc) logical instance

See [Polling Functions](#) for details on all the polling functions.



3.3 Operating System Support

The software package supports the Linux*, FreeBSD* and Windows* operating systems. Intel® QuickAssist Technology software requires that the following crypto modules be present on the system: `sha256-generic.ko` and `sha512-generic.ko`. The Acceleration driver is validated with the Linux operating system only. Details of the specific operating system versions supported depend on the release version. See the Release Notes for your release version for details on the specific operating system support provided in that release version.

3.4 OpenSSL* Library Inclusion and Usage

The Intel® Communications Chipset 8900 to 8920 Series Linux* package is distributed with an OpenSSL library file. This library file has certain dependencies that will be met in most cases. In the event that these dependencies are not met, it may be necessary to build OpenSSL on the development platform and link any ProductNameShort applications to the relevant OpenSSL library.

3.5 Support for Multiple Acceleration Hardware Generations

Note: Not all Intel® QuickAssist Technology releases come with support for multiple acceleration hardware generations.

Note: See [Utility for Loading Configuration Files and Sending Events to the Driver - adf_ctl](#) on page 34 for additional details.

Software Architecture

The acceleration drivers for Intel® Communications Chipset 8900 to 8920 Series and Intel® Communications Chipset 8925 to 8955 Series devices are not compatible, however later Intel® QuickAssist Technology software releases allow for both sets of drivers to be loaded on the same target. Compatibility with the Intel® QuickAssist Technology API is maintained via a "mux" layer that provides the dynamic linking to the appropriate driver based on the particular device.

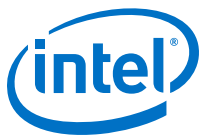
Software Packaging

This package includes:

- QAT 1.5 tarball of Intel architecture (IA) driver
- QAT 1.6 tarball of IA driver
- `qat_mux` (included in the QAT 1.6 tarball), which exposes the Intel® QuickAssist Technology API in the case where both above drivers are installed. When only one of the above drivers is installed, the Intel® QuickAssist Technology API is exposed by the driver and the `qat_mux` is not installed.

Different devices are supported by different Intel® QuickAssist Technology drivers; please see the following table:

Device	Driver
DH8900 - DH8920	QAT 1.5
C2XXX	QAT 1.5
DH8925 - DH8955	QAT 1.6



In the Intel® QuickAssist Technology software package, the directory "QAT1.5" contains the driver for the Intel® Communications Chipset 8900 to 8920 Series and Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure devices, and the directory "QAT1.6" contains the driver for the Intel® Communications Chipset 8925 to 8955 Series devices. The "mux" directory contains the software to build in support for all of the above devices.

Build Installation Details

Some Intel® QuickAssist Technology releases can support multiple acceleration hardware generations (e.g., both Intel® Communications Chipset 8900 to 8920 Series and Intel® Communications Chipset 8925 to 8955 Series). By default, software releases with support for multiple acceleration hardware generations will build or install according to the devices visible on the platform. For instance:

- If one or more Intel® Communications Chipset 8900 to 8920 Series devices are visible on the PCIe bus and no Intel® Communications Chipset 8925 to 8955 Series device is present, the installer.sh will build with support for Intel® Communications Chipset 8900 to 8920 Series devices only.
- If one or more Intel® Communications Chipset 8925 to 8955 Series devices are visible on the PCIe bus and no Intel® Communications Chipset 8900 to 8920 Series device is present, the installer.sh will build with support for Intel® Communications Chipset 8925 to 8955 Series devices only.
- If one or more Intel® Communications Chipset 8925 to 8955 Series devices are visible on the PCIe bus and one or more Intel® Communications Chipset 8900 to 8920 Series devices are present, the installer.sh will build with support for both Intel® Communications Chipset 8900 to 8920 Series devices and Intel® Communications Chipset 8925 to 8955 Series.

There are two primary usage models for building with support for multiple acceleration hardware generations:

1. Concurrent usage of acceleration devices across multiple acceleration hardware generations.
2. Deployment of a software release/image that supports multiple acceleration hardware generations, without the expectation that a given platform will have more than one acceleration hardware generation present.

To support multiple acceleration hardware generations, the icp_qa_al.ko kernel module is not used. Instead, a "mux" kernel module (qat_mux.ko) and one or both of qat_1_5_mux.ko and qat_1_6_mux.ko (depending on which hardware must be supported) are used. In addition, any applications that make use of the acceleration software must link to different libraries. In summary, the following table applies:

Case	Kernel object(s)	User Space object(s)	Static Libraries
QAT 1.5 only build option	icp_qa_al.ko	libicp_qa_al_s.so	libicp_qa_al.a
QAT 1.6 only build option	icp_qa_al.ko	libicp_qa_al_s.so	libicp_qa_al.a
QATmux case supporting multiple acceleration hardware generations	qat_1_5_mux.ko qat_1_6_mux.ko qat_mux.ko	libqat_1_5_mux_s.so libqat_1_6_mux_s.so libqat_mux_s.so	libqat_1_5_mux.a libqat_1_6_mux.a libqat_mux.a

User space applications in a mux installation should link against libqat_mux_s.so or libqat_mux.a; there's no need to link against the other build objects.



Part 2: Acceleration Drivers



4.0 Acceleration Drivers Overview

The Intel® Communications Chipset 8900 to 8920 Series contains:

- Acceleration Drivers - These drivers are described in this chapter.

For each supported acceleration service (Cryptographic, Data Compression), the following application usage models are supported:

- Kernel mode, where both the application and the service(s) are running in kernel space.
- Direct user space access to services running in user space. In this model, both the application and service(s) are running in user space and access to the hardware is also performed from user space. The kernel space driver is needed to perform the mapping for user space access.

The Acceleration Drivers are supported on 64-bit and 32-bit kernels. 32-bit user space applications are supported on 32-bit and 64-bit kernels.

For Linux*, the acceleration drivers are provided for both user and kernel space. A porting guide is available that provides guidance on porting the software to other Operating Systems including RTOSs that do not distinguish between user and kernel space. Refer to the *Intel® QuickAssist Technology Acceleration Software OS Porting Guide* for additional information.

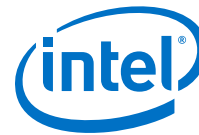
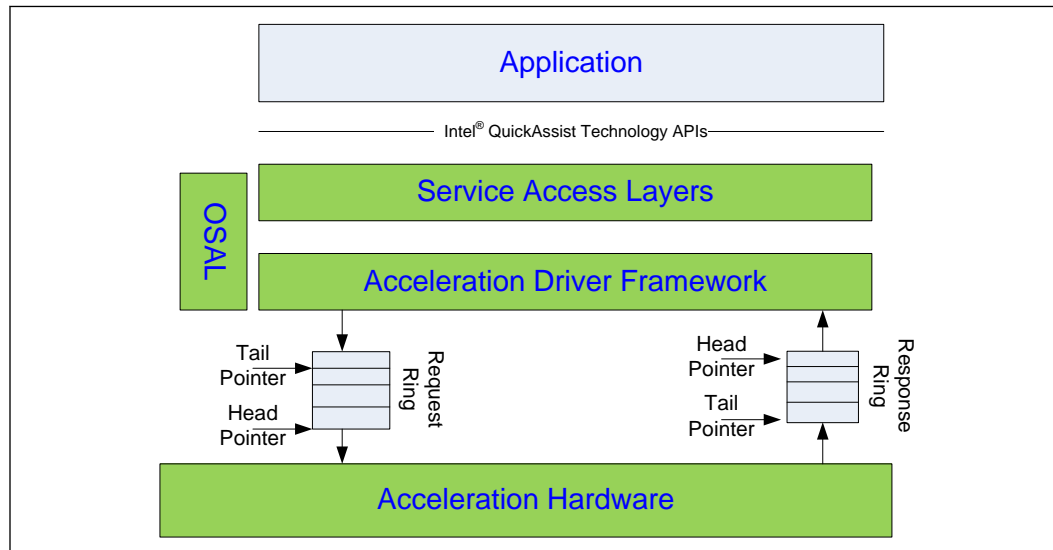
4.1 Hardware Assisted Rings

Hardware assisted rings are used as the communication mechanism to transfer requests between the CPU and the accelerator(s) on the chipset device and vice-versa. The hardware supports 256 rings, each with head and tail Configuration Status Register (CSR) pointers that are mapped to PCIe* memory on the CPU. The rings may be configured as:

- Request rings, where the CPU is a producer and the accelerator is a consumer
- Response rings, where the accelerator is a producer and the CPU is a consumer

The CPU may be arranged as a producer or a consumer on a ring, but cannot be both a consumer and producer on the same ring, as shown in the following figure. This is to avoid atomicity issues associated with multiple writers.

Note: The rings are configured and serviced by the provided kernel space driver for use by the application either in kernel or user space.

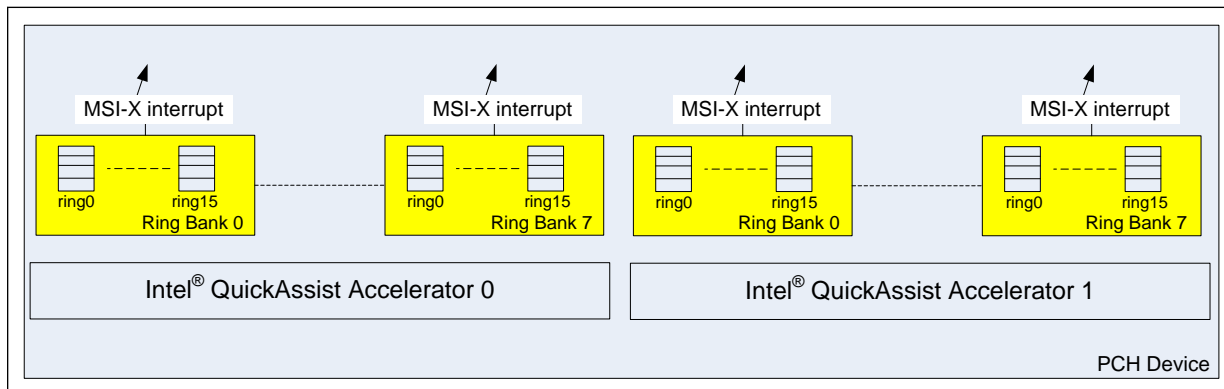

Figure 6. Intel® QuickAssist Accelerator Ring Access


Rings are grouped into *ring banks* with each ring bank containing 16 rings, and there are 8 ring banks for each accelerator.

For each ring bank, hardware supports the generation of the interrupt when data is available for processing on the response ring within the bank.

On each accelerator in the chipset device, there are eight independent ring banks. Each ring bank has an associated ring interrupt. If the OS supports MSI-X interrupts, the response may be directed to any core on system. This allows an even distribution of response processing among the cores on the system. The configuration of bank interrupts and core affinity is detailed in [Affinity Parameters](#) on page 154.

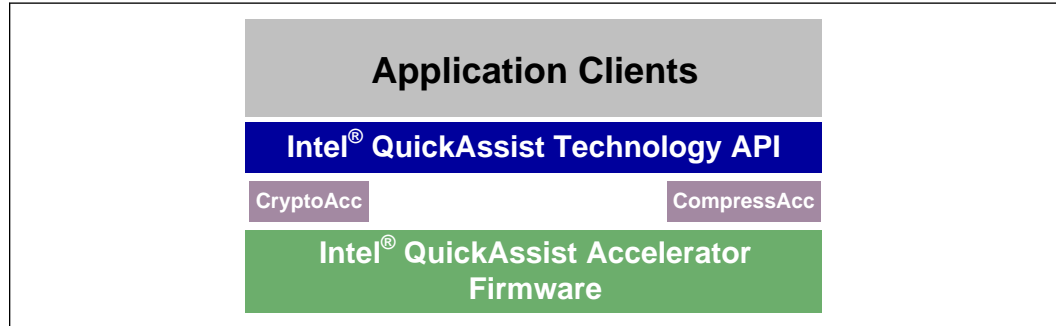
Depending on the chipset device model number, there are up to two accelerators on the device. The following figure shows an overview of the rings, ring banks and accelerators for a single chipset.

Figure 7. Ring Partitioning on the Chipset Device


4.2 Basic Software Context for Acceleration Drivers

The following figure depicts the basic OS-agnostic software model for the acceleration drivers.

Figure 8. Basic Software Context



The key elements of this model are as follows:

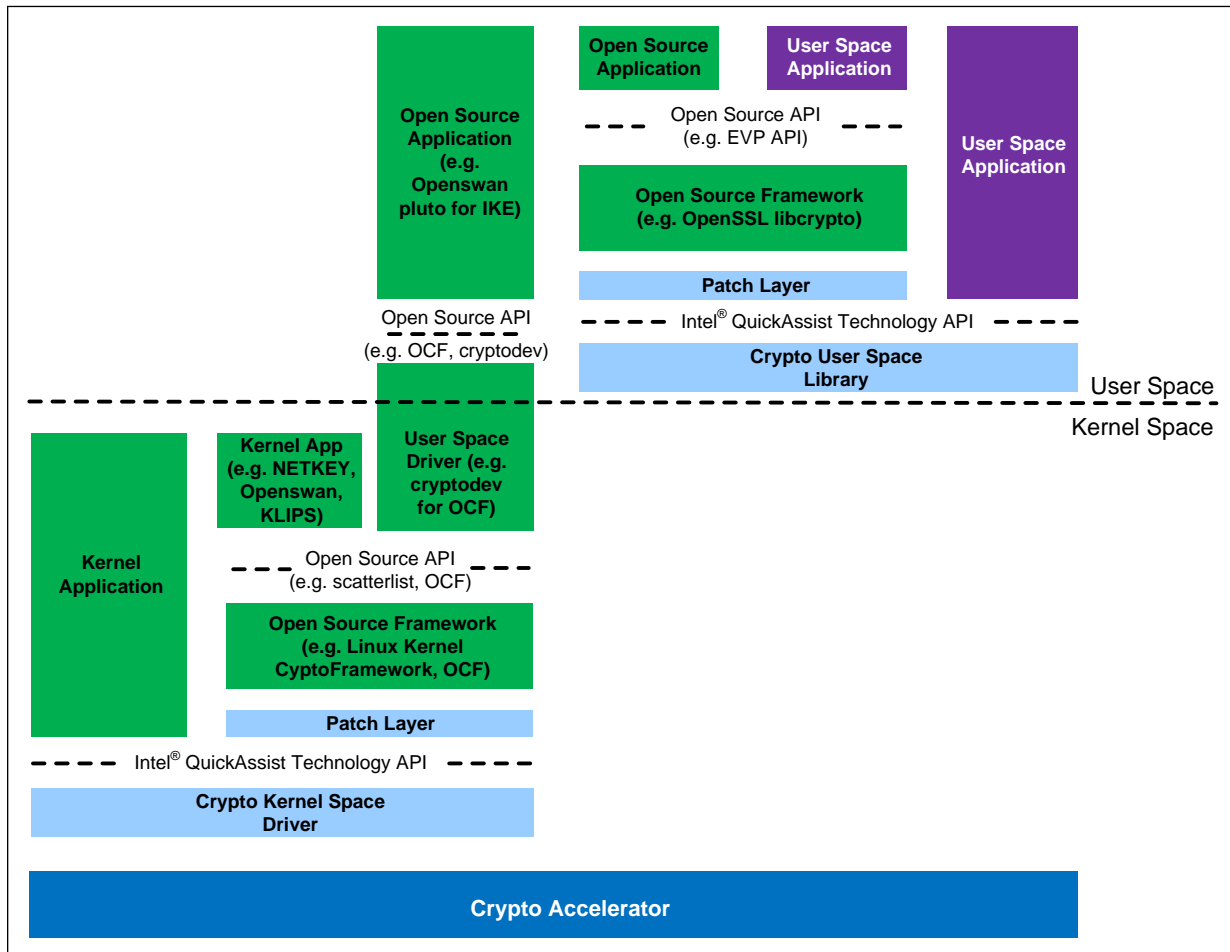
- The firmware encompasses software executing on the accelerator(s).
- Intel® architecture software entities that fall into two groups:
 - Driver level entities - CryptoAcc, CompressAcc, and the Intel® QuickAssist Technology API
 - Application level entities - application clients
- Application-level software that runs on Intel® architecture.
 - Application entities executing at an Intel® architecture level that make use of the accelerators via the Intel® QuickAssist Technology APIs.

4.3 Linux* Software Context for Acceleration Drivers

The following figure shows an example of the Linux* operating environment for the Acceleration Driver Framework.



Figure 9. Linux Software Context



The Services support applications in kernel space as well as user space. User space access is hardware direct access with mapping from kernel space driver. Catering for these access options provides full flexibility in the use of the accelerator.

The driver architecture supports simultaneous operation of multiple applications using any and all combinations of acceleration access options. However, some limitations apply. These are called out clearly in following topics.

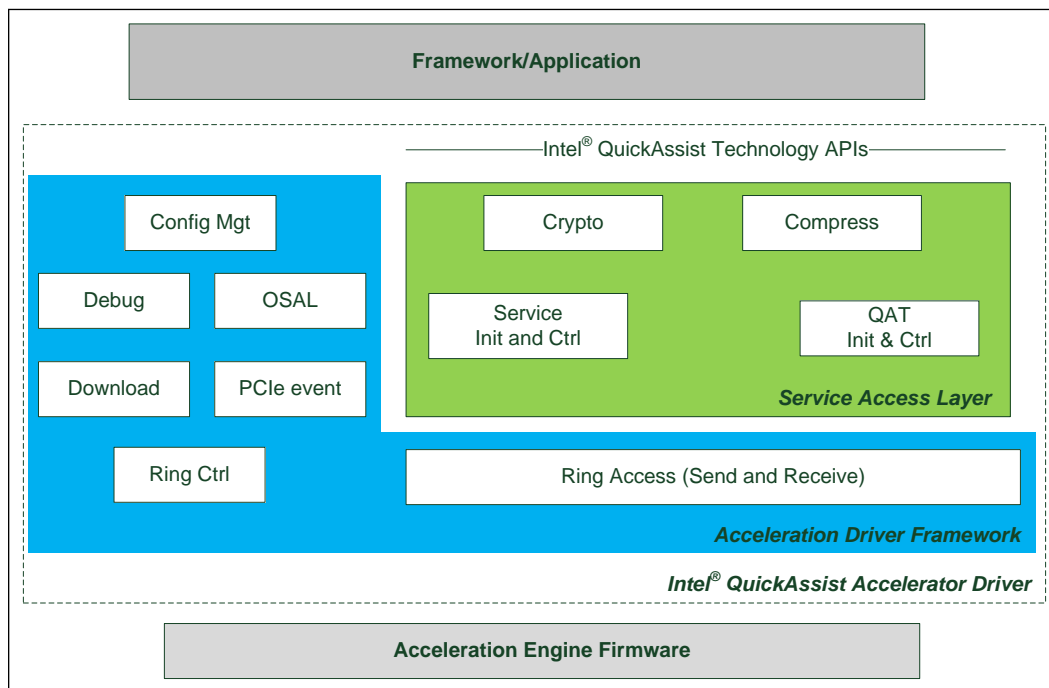
Note: The applications identified in the figure above are examples only and do not serve as a statement of intent for enabling.

Note: Software packages for patches, such as OpenSSL, Linux Kernel Crypto Framework, and NetKey and zlib are distributed separately. See [Product Documentation](#) on page 14. You will need an Intel Business Link (IBL) account and a subscription to the Electronic Design Kit (EDK).

4.4 Acceleration Drivers

The Acceleration Driver is divided into a number of functional components as shown in the following figure. The figure shows the basic driver framework.

Figure 10. Acceleration Driver Framework



4.4.1 Framework Overview

An acceleration driver contains a number of logical units that are primarily exposed via the Intel® QuickAssist Technology APIs. Figure 10 on page 32 depicts the main components of the driver. These are:

- **Service Access Layer (SAL)**

Provides the main access to the acceleration services of the accelerator. Each service is provided by a service entity in that layer. Though contained in a single logical layer, each service is separate and distinct and as such services do not depend on each other.

- **Acceleration Driver Framework (ADF)**

An acceleration driver provides a supporting framework which contains services that the SAL depends on and also provides the hardware level interactions for PCI in particular, including PCI registration and interaction.

4.4.2 Service Access Layer

The Service Access Layer (SAL) is responsible for providing access to the individual acceleration services contained in the accelerator. As shown in Figure 10 on page 32, the layer is made up of the individual services as well as an Initialization and Control component.

This layer is largely OS-agnostic. In particular, the layer is designed in such a way as to allow it to operate in kernel space as well as user space Linux* environments.

The primary responsibilities of this layer are as follows:



- Register for notification of, query, observe and handle initialization/discovery/error events from the ADF framework. The layer initializes and stops services based on the state of the accelerator as indicated by ADF.
- Initialize the service layers based on the settings in a configuration file.
- Initialize and model the logical accelerator instances as configured in the configuration file.
- Be aware of the execution context for the SAL, that is, whether operating as a driver in kernel space or a library in user space and perform the necessary initializations required.
- Process Intel® QuickAssist Technology API functions and pass them on as requests to the firmware.

4.4.3 Acceleration Driver Framework

This topic outlines the services in the ADF that the SAL depends on.

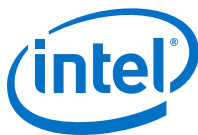
Services include:

- **Events:** The SAL relies on the ADF for an event notification function with which the SAL registers to get notified of key runtime events. It uses these events to trigger initialization and shutdown operations in particular. The SAL also queries the ADF for the status.
- **Discovery:** The ADF framework is responsible for all hardware level discovery and provides notification to the SAL when accelerator discovery events occur such as accelerator plug and play events.
- **Download & Init:** The ADF framework takes care of the download and starting of the firmware. The ADF notifies the SAL that the firmware is downloaded and started.
- **Ring Control and Access:** The ADF provides the mechanism by which the accelerator rings are configured, including the enabling of interrupts on ring sets. In addition, the ADF abstracts the communication mechanism with the accelerator.
- **Configuration:** ADF provides access to the configuration text files used to configure an acceleration driver. Some elements of the configuration file such as ring bank configuration belong to the ADF itself, while other settings are owned by the SAL. The ADF provides the mechanism by which the SAL gets access to the configuration settings.
- **OS Abstraction:** The SAL layer is OS independent and makes use of the OSAL provided as part of the ADF.

Note: When operating in user space, the SAL should be considered to have the same dependencies on the ADF as it does in kernel space.

4.4.4 Acceleration Driver Configuration File

An acceleration driver has a configuration file that is used to configure the driver for runtime operation. There is a single configuration file for each PCH device in the system. The configuration file format is described in [Acceleration Driver Configuration File](#) on page 69. The older legacy configuration file format (which is still supported) is described in [Acceleration Driver Configuration File - Earlier File Format](#) on page 149.



4.4.5 Utility for Loading Configuration Files and Sending Events to the Driver - `adf_ctl`

The `adf_ctl` user space utility is separate to the driver and provides the mechanism for:

- Loading configuration file data to the kernel driver. The kernel space driver uses the data and also provides the data to the user space driver.
- Sending events to the driver to bring devices up and down.

The `adf_ctl` utilities provided in the QAT 1.5 package and earlier QAT 1.6 packages can only be used to interface with the driver they are provided with.

The `adf_ctl` provided with the QAT1.6 driver in the single package can be used to interface with both drivers. It can bring up all devices supported by both drivers.

Usage

`./adf_ctl [dev] [up|down|reset]` - to bring up or down or reset device(s).

or

`./adf_ctl status` - to print device(s) status

Device Enumeration

Device enumeration varies within the driver code, in `adf_ctl` and on the API. This is best illustrated with an example. The following table illustrates device enumeration on a platform with three different device types, two DH895xccc, two DH89xxccc and one C2xxx.

Table 1. Device Enumeration Example

Driver	adf_ctl status			Conf File Name	API	
	devices	types	Inst_id		Used by client in call to <code>icp_sal_poll</code> Bank, etc.	Passed by mux to driver in call to <code>icp_sal_poll</code> Bank, etc
	<i>accelId</i>	<i>hw_data.dev_class.name</i>	<i>hw_data.InstanceId</i>		<i>accelId on API</i>	<i>accel_dev.accelId in driver</i>
QAT1.6	icp_dev0	dh895xcc	0	dh895xcc_qa_dev0.conf	0	0
QAT1.6	icp_dev1	dh895xcc	1	dh895xcc_qa_dev1.conf	1	1
QAT1.5	icp_dev2	dh89xxcc	0	dh89xxcc_qa_dev0.conf	2	0
QAT1.5	icp_dev3	c2xxx	0	c2xxx_qa_dev0.conf	3	1
QAT1.5	icp_dev4	dh89xxcc	1	dh89xxcc_qa_dev1.conf	4	2

Examples of Manual Sequence for Starting the Driver



Note: For the full installation, see the *Intel® Communications Chipset 89xx Series Software for Linux* Getting Started Guide*.

Case where only DH895xcc devices are on the platform

1. Copy firmware to `/lib/firmware/dh895xcc`
2. Copy a config file for each device to `/etc`
3. `insmod ./QAT1.6/build/icp_qa_al.ko`
4. `./QAT1.6/build/adf_ctl up`

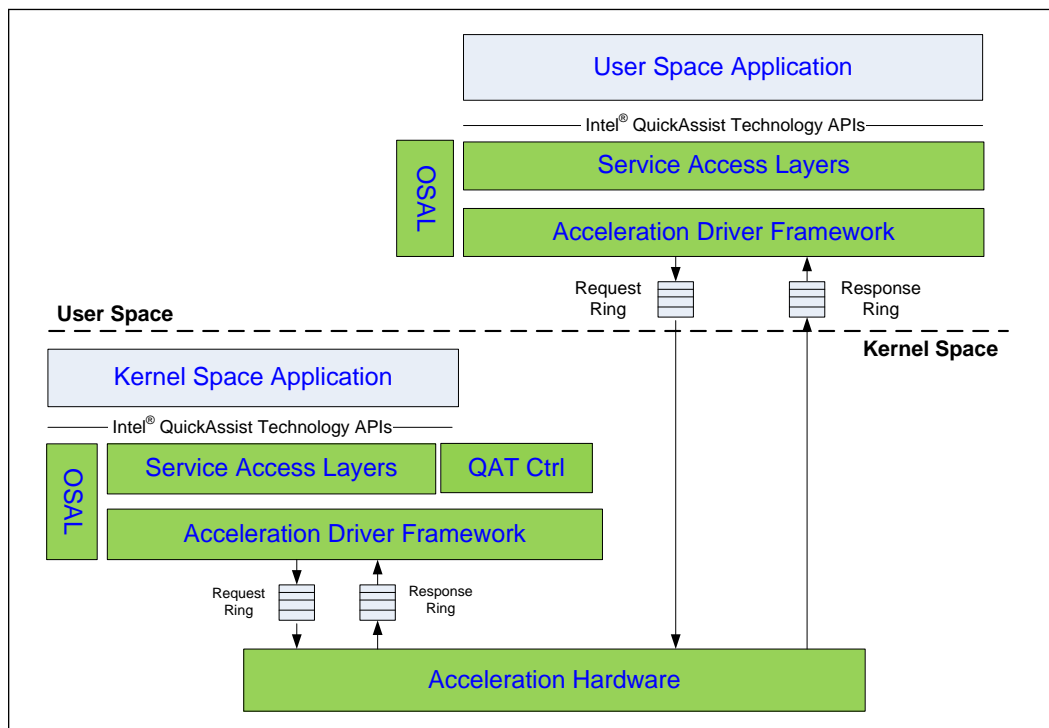
Case where DH895xcc and DH89xxcc devices are on the platform

1. Copy firmware for DH89xxcc to `/lib/firmware` and for DH895xcc to `/lib/firmware/dh895xcc`
2. Copy a config file for each device to `/etc`
3. `insmod ./QAT1.6/build/qat_mux.ko`
4. `insmod ./QAT1.5/build/qat_1_5_mux.ko`
5. `insmod ./QAT1.6/build/qat_1_6_mux.ko`
6. `./QAT1.6/build/adf_ctl up`

4.5 Acceleration Architecture in Kernel and User Space

The Intel® QuickAssist Accelerator software is architected to allow it operate in either kernel or user space using a “build time” decision. The overall architecture of the software stack is shown in the following figure.

Figure 11. Software Architecture for Kernel and User Space



The Intel® QuickAssist Technology API is OS agnostic and has the same function signatures in both kernel or user space. The SAL component is also OS agnostic and may be compiled as a user space library or as a kernel space module. The SAL uses the OSAL for all OS services and versions of OSAL have been implemented for Linux user space and kernel space.

4.5.1 User Space Memory Allocation

For user space applications, two aspects of memory allocation need to be considered:

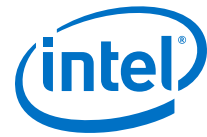
- Accelerator driver memory allocation
- Application payload memory allocation

4.5.1.1 Accelerator Driver Memory Allocation

At initialization, the accelerator driver allocates memory for use in communications with the Intel® QuickAssist Accelerator hardware. This memory needs to be resident, DMA accessible and needs a physical address to provide to the accelerator hardware.

In kernel space, the SAL calls the OSAL memory routines to allocate this memory. Principally, the function used by SAL is `osalMemAllocContiguousNUMA`. In the kernel, this OSAL routine is implemented with `kmalloc_node`. Memory allocated using `kmalloc_node` is guaranteed to be contiguous, resident and the OSAL routine also exists to retrieve the associated physical address.

In user space, it is a little more complex. The OSAL implementation of `osalMemAllocContiguousNUMA` needs to return memory that is resident and contiguous. To do this, the OSAL in kernel space creates a device, called



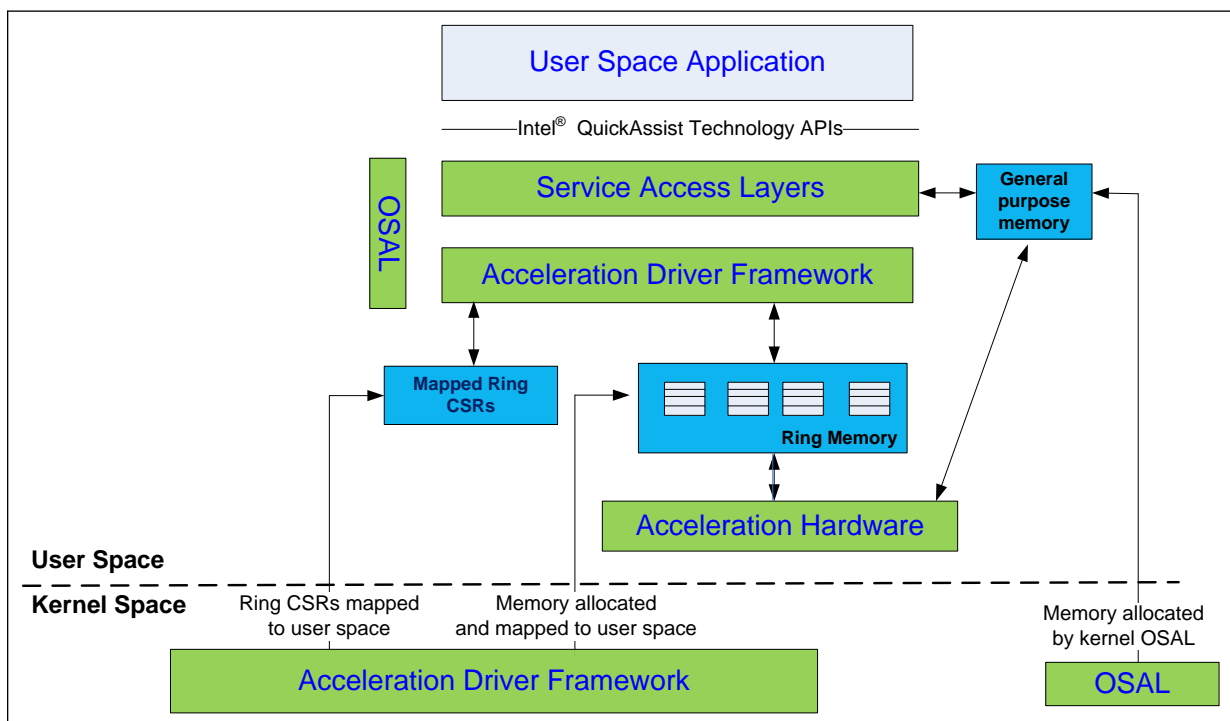
`icp_dev_mem` that may be called through an `IOCTL` function by the OSAL in user space to allocate memory. When called with `IOCTL_DEV_MEM_IOC_MEMALLOC`, the OSAL kernel mode driver returns the allocated memory.

For communications with the Intel® QuickAssist Accelerator device, the ADF needs access to the rings. The hardware ring CSRs are mapped from kernel space MMIO space to the application's user space by ADF. The DRAM memory for the hardware rings are also mapped to the user space application. In user space, the ADF exposes a ring put and a ring get API to the SAL to allow it to communicate with the Intel® QuickAssist Accelerator hardware.

The following figure shows the ring CSRs and allocation buffers that are required to be mapped to user space.

Note: If your software has another mechanism for the allocation of contiguous memory, for example, by reserving an area of memory from the OS, then replace the OSAL memory functions (see `$ICP/quickassist/utilities/osal/include/Osal.h` for details) with your specific implementation.

Figure 12. User Space Memory Allocation at Initialization



4.5.1.2 Application Payload Memory Allocation

When performing offload operations through the Intel® QuickAssist Technology API, it is required that the payload data be placed in a buffer that is resident, physically contiguous and is DMA accessible from the acceleration hardware. It is the application's responsibility to provide buffers with these constraints. A scheme similar to the OSAL implementation mentioned above may be implemented by the user space application.



Buffers are passed to the Intel® QuickAssist Accelerator service access layer with virtual addresses. However, the accelerator layers need to pass physical addresses to the hardware, therefore a virtual-to-physical address translation is required. The Intel® QuickAssist Technology API allows an application to register a function that will do this virtual-to-physical translation.

Cryptographic service	<code>cpaCySetAddressTranslation</code>	See the <i>Intel® QuickAssist Technology Cryptographic API Reference Manual</i> for details.
Data Compression service	<code>cpaDcSetAddressTranslation</code>	See the <i>Intel® QuickAssist Technology Data Compression API Reference Manual</i> for details.

When the SAL requires the physical address, it calls the registered function.

Note: This address translation function is called at least once per request. Consequently, for optimal performance, the implementation of this function should be optimized.

4.5.2 User Space Additional Functions

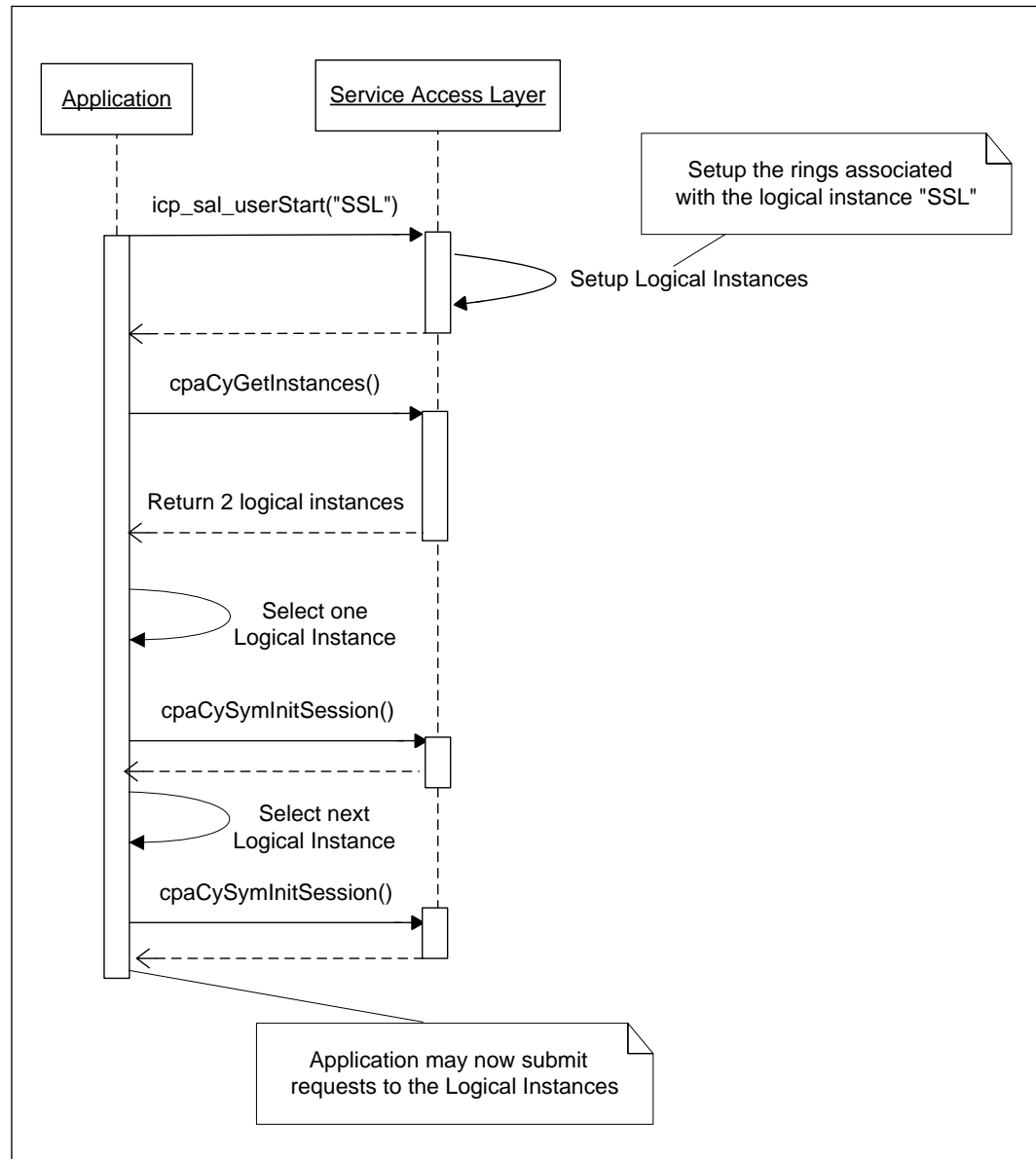
To allow a user space process access to the Intel® QuickAssist Accelerator rings, the service access layer needs to be configured to expose logical instances to the user space process. Logical instances are configured using the per device configuration file. See [User Space Configuration](#) on page 39 for an example.

To allow each process to have separate logical instances, the configuration file groups a set of logical instances by name. The process then needs to call the [icp_sal_userStartMultiProcess](#) on page 130 function (or [icp_sal_userStart](#) on page 129 if the older configuration file format is used) at initialization time with the name associated with the group of logical instances. Similarly, on process exit, to free the resources and make them available to other processes with the same name, the process needs to call the function [icp_sal_userStop](#) on page 131.

For example, in the sequence in the following figure, the user has configured the Service Access Layer to have two crypto logical instances available for the process called "SSL". The user space process may then access these logical instances by calling the `cpaCyGetInstances` function. The application may then initiate a session with these logical instances and perform a cryptographic operation. See the *Intel® QuickAssist Technology Cryptographic API Reference Manual* for more information on the API functions available for use.



Figure 13. User Space Process with Two Logical Instances



4.5.3 User Space Configuration

The section of the configuration file that details user space configuration follows the `[KERNEL]` section.

For example, in the sequence in [Figure 13](#) on page 39, the user has configured the service access layer to have two crypto logical instances available for the process called "SSL".



For this example, the logical instances section of the configuration file is as follows:

```
[KERNEL]
NumberCyInstances = 0
NumberDcInstances = 0

[SSL]
NumberCyInstances = 2
NumberDcInstances = 0
NumProcesses = 1

# Crypto - User instance #0
Cy0Name = "SSL0"
Cy0IsPolled = 1
Cy0AcceleratorNumber = 0,1
# List of core affinities
Cy0CoreAffinity = 0,1

# Crypto - User instance #1
CylName = "SSL1"
CylIsPolled = 1
CylAcceleratorNumber = 2,3
# List of core affinities
CylCoreAffinity = 2,3
```

In this example, the user process SSL configures two logical instances (called "SSL0" and "SSL1"), each of which targets specific acceleration units, so that load balancing among the four (assuming the top SKU) acceleration units is achieved.

4.5.4 User Space Response Processing

As in the case of kernel space operation, there are two modes of response processing for user space operation:

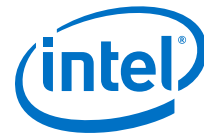
- Polled mode
- Interrupt mode

4.5.4.1 User Space Interrupt Mode

Note: User space interrupt mode is being removed from future Intel® QuickAssist Technology releases. A new event-based user space notification mechanism will be added. Please discuss any concerns with your Intel representative.

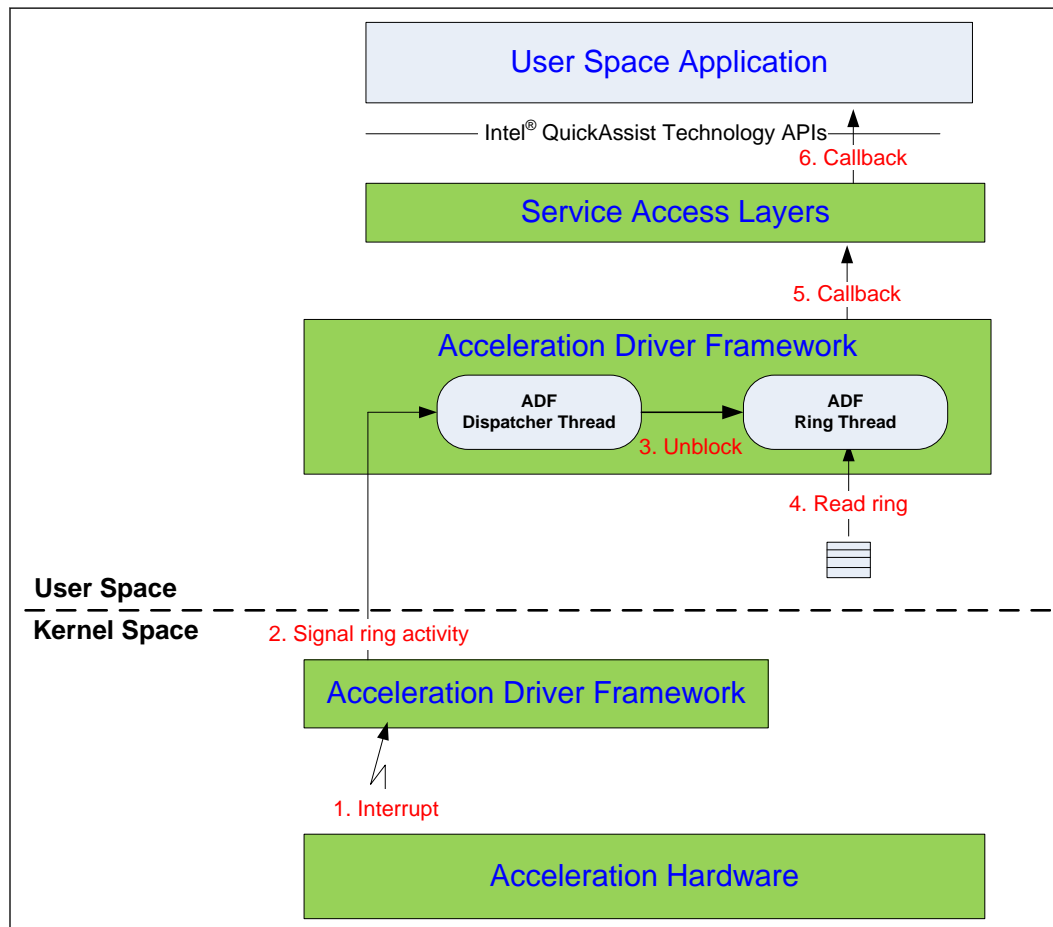
Response ring processing in interrupt mode differs slightly from the kernel mode response ring processing since the user space application needs to be signaled when a response is placed on the response ring by the Intel® QuickAssist Accelerator hardware.

The ADF is responsible for managing this signaling path. Initially, user space ADF creates a dispatcher thread that is responsible for handling the notifications from the ADF in kernel space. Upon creation, this thread blocks on reading a Linux character device until the dispatcher thread has been signaled by the ADF in kernel space. For each user space response ring that is subsequently created, ADF creates a ring thread in user space for reading the response ring.



Upon receiving a response, the ADF in kernel space shall post a signal to wake-up the blocked dispatcher thread. The dispatcher thread notifies the relevant ring thread and the ADF will read the contents of the ring in the context of this ring thread. The ADF calls back SAL and SAL in turn calls back the application to signal the completion of the original request. This sequence is depicted in the following figure.

Figure 14. User Space Response Processing for Interrupt Mode



4.5.4.2 User Space Polled Mode

The sequence for user space polling does not differ from that described in [Polled Mode](#) on page 24.

4.6 Managing Acceleration Devices Using `qat_service`

The `qat_service` script is installed with the software package in the `/etc/init.d/` directory. The script allows a user to start, stop, or query the status (up or down) of a single device or all devices in the system.

Usage:

```
# ./qat_service start||stop||status||restart||shutdown
```



To view all devices in the system, use:

```
# ./qat_service status
```

If there are two acceleration devices in the system for example, the output will be similar to the following:

```
icp_dev0 is up
icp_dev1 is up
```

For a system with multiple devices, you can start, stop or restart each individual device by passing the device to be restarted or stopped as a parameter (*icp_dev<N>*). For example:

```
# ./qat_service stop icp_dev0
```

where the device number *<N>* is equal to 0 in this case.

The *shutdown* qualifier enables the user to bring down all devices and unload driver modules from the kernel. This contrasts with the *stop* qualifier which brings down one or more devices, but does not unload kernel modules, so other devices can still run.

4.7 Intel® QuickAssist Technology Entries in the /proc Filesystem

For kernel space instances, the following /proc filesystem entries are created to provide information on the driver and APIs, provided the related entry has been enabled in the drivers configuration file.

/proc/ icp_dh89xcc_devX/ files, where X is the device number	Description of information contained in that file
./cfg_debug	Internal configuration table generated from: /etc/dh89xcc_qa_devX.conf and from some internal data, e.g., firmware version. It is useful to check which user processes and instances have been configured.
./qat	Statistics for Intel® QuickAssist Technology (QAT), overall number of requests/responses per ME. FW is loaded on each ME, if ME 0 gets one request, processes it and put it back on the ring, then the FW counters for Request and Response will be incremented by 1 for that ME. Example output for one ME is: +-----+ Statistics for Qat Instance 0 +-----+ Firmware Requests[AE 0]: 1 Firmware Responses[AE 0]: 1 For QAT 1.5 and QAT 1.6, this also triggers the heartbeat query below.
./version	Lists hardware, software and API versions in use. Example output for QAT1.6: +-----+ Hardware and Software versions for device 0 +-----+ Hardware Version: A0 SKU2 Firmware Version: 2.2.0

continued...



/proc/ icp_dh89xcc_devX/ files, where X is the device number	Description of information contained in that file
	<pre> MMP Version: 1.0.0 Driver Version: 2.2.0 Lowest Compatible Driver: 2.0 QuickAssist API CY Version: 1.8 QuickAssist API DC Version: 1.3 +-----+ </pre> <p>'Lowest Compatible Driver' indicates the lowest QAT driver version that this driver is compatible with in a virtualized system, where one driver is on the Host and the other is in a Guest.</p>
./cy/IPSecY	
./dc/IPCompY	For cy and dc stats, see Section 4.7 and Section 5.2.2
./et_ring_ctrl/bank_Y/conf	<p>Refers to EagleTail_Ring_Control, this conf file gives a summary on all EagleTailRings in use in bank_Y, where Y is one of the banks configured for use.</p> <p>Example output:</p> <pre> cat /proc/icp_dh895xcc_dev0/et_ring_ctrl/bank_0/conf ----- Bank 0 Configuration ----- Interrupt Coalescing Enabled Interrupt Coalescing Counter = 10000 Interrupt mask: 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 User interrupt mask: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 Polling mask: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 Coalesc reg: 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 Bank empty stat: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 Bank nempty stat: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ----- Rings: Ring Number: 0, Config: 80000006, Base Addr: ffff880267e50000 Head: 0, Tail: 0, Space: 1000, inflight: 0, Name: Cy0RingAsymTx Ring Number: 2, Config: 8000000a, Base Addr: ffff88021ea60000 Head: 0, Tail: 0, Space: 10000, inflight: 0, Name: Cy0RingSymTx Ring Number: 4, Config: 8000000a, Base Addr: ffff88021e8a0000 Head: 0, Tail: 0, Space: 10000, inflight: 0, Name: Cy0RingNrbgTx Ring Number: 6, Config: 8000000a, Base Addr: ffff88021ffd0000 Head: 0, Tail: 0, Space: 10000, inflight: 0, Name: Dc0RingTx Ring Number: 8, Config: 5405, Base Addr: ffff880267e51000 Head: 0, Tail: 0, Space: 1000, inflight: 0, Name: Cy0RingAsymRx Ring Number: 10, Config: 5408, Base Addr: ffff880220140000 Head: 0, Tail: 0, Space: 4000, inflight: 0, Name: Cy0RingSymRx Ring Number: 12, Config: 5408, Base Addr: ffff8802200cc000 Head: 0, Tail: 0, Space: 4000, inflight: 0, Name: Cy0RingNrbgRx Ring Number: 14, Config: 5408, Base Addr: ffff8802202b4000 Head: 0, Tail: 0, Space: 4000, inflight: 0, Name: Dc0RingRx ----- </pre>
./et_ring_ctrl/bank_Y/ ring_Z	<p>Gives information on each specific ring. For example ring_0 from the above conf entry will give the data on that ring and accelerator number associated with it in addition to the information given in the conf entry:</p> <pre> ----- Ring Configuration ----- Service Name: Cy0RingAsymTx Accelerator Number: 0, Bank Number: 0, Ring Number: 0 Ring Config: 80000006 Tx, Base Address: ffff880267e50000, Head: 0, Tail: 0, Space: 1000 Message size: 64, Max messages: 64, Current messages: 0 Ring Empty flag: 1, Ring Nearly Empty flag: 1 </pre>



<code>/proc/ icp_dh89xcc_devX/ files, where X is the device number</code>	Description of information contained in that file
	----- Ring Data ----- Memory Address: <Contents of memory address (64bytes)>

4.8 Debug Feature

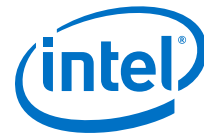
For user space applications, there are a number of Intel® QuickAssist Technology API functions that enable a user to retrieve statistics for a service instance. These functions include:

- `cpaCyDhQueryStats64` - Query statistics (64-bit version) for Diffie-Hellman operations.
- `cpaCyDsaQueryStats64` - Query 64-bit statistics for a specific DSA instance.
- `cpaCyKeyGenQueryStats64` - Queries the Key and Mask generation statistics (64-bit version) specific to an instance.
- `cpaCyPrimeQueryStats64` - Query prime number statistics specific to an instance.
- `cpaCyRsaQueryStats64` - Query statistics (64-bit version) for a specific RSA instance.
- `cpaCySymQueryStats64` - Query symmetric cryptographic statistics (64-bit version) for a specific instance.
- `cpaCyEcQueryStats64` - Query statistics for a specific EC instance.
- `cpaCyEcdhQueryStats64` - Query statistics for a specific ECDH instance.
- `cpaCyEcdsaQueryStats64` - Query statistics for a specific ECDSA instance.
- `cpaCyDrbgQueryStats64` - Returns statistics specific to a session, or instance, of the RBG API.
- `cpaDcGetStats` - Retrieves the current statistics for a compression.

See the *Intel® QuickAssist Technology Cryptographic API Reference Manual* and the *Intel® QuickAssist Technology Data Compression API Reference Manual* for detailed information.

For kernel space instances, the same information can be obtained from the `/proc` file system if the required statistics parameters are enabled in the configuration file, as the following configuration file extract shows. See also [Statistics Parameters](#) on page 73 for more detail.

```
#Statistics, valid values: 1,0
statsGeneral = 1
statsDc = 1
statsDh = 1
statsDrbg = 1
statsDsa = 1
statsEcc = 1
statsKeyGen = 1
statsLn = 1
```



```
statsPrime = 1
statsRsa = 1
statsSym = 1
```

For each instance, a file is created with a name that is the same as the instance name specified in the configuration file. For example, if in the “User Process Instance Section” of the configuration file, the IPSec0, IPSec1, IPSec2 and IPSec3 names are used, the following command gives the result:

```
# ls -l /proc/icp_dh89xxcc_dev0/cy

total 0

-r----- . 1 root root 0 Apr 18 13:48 IPSec0
-r----- . 1 root root 0 Apr 18 13:48 IPSec1
-r----- . 1 root root 0 Apr 18 13:48 IPSec2
-r----- . 1 root root 0 Apr 18 13:48 IPSec3
```

The statistics can then be queried simply by running `cat` on the corresponding file in the `/proc` file system. For example:

```
# cat /proc/icp_dh89xxcc_dev0/cy/IPSec0
```

The output is similar to the following:

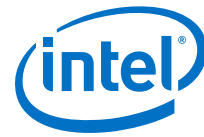
```
+-----+
| Statistics for Instance                               IPSec0 |
| Symmetric Stats                                         |
+-----+
| Sessions Initialized:                                   86 |
| Sessions Removed:                                       86 |
| Session Errors:                                          0 |
+-----+
| Symmetric Requests:                                     960 |
| Symmetric Request Errors:                               0 |
| Symmetric Completed:                                    960 |
| Symmetric Completed Errors:                             0 |
| Symmetric Verify Failures:                             0 |
+-----+
| DSA Stats                                               |
+-----+
| DSA P Param Gen Requests-Succ:                         0 |
| DSA P Param Gen Requests-Err:                          0 |
| DSA P Param Gen Completed-Succ:                        0 |
| DSA P Param Gen Completed-Err:                         0 |
+-----+
| DSA G Param Gen Requests-Succ:                         1 |
| DSA G Param Gen Requests-Err:                          0 |
| DSA G Param Gen Completed-Succ:                        1 |
| DSA G Param Gen Completed-Err:                         0 |
+-----+
| DSA Y Param Gen Requests-Succ:                        20 |
| DSA Y Param Gen Requests-Err:                          0 |
| DSA Y Param Gen Completed-Succ:                        20 |
| DSA Y Param Gen Completed-Err:                         0 |
+-----+
| DSA R Sign Requests-Succ:                              0 |
| DSA R Sign Request-Err:                                0 |
| DSA R Sign Completed-Succ:                             0 |
| DSA R Sign Completed-Err:                              0 |
+-----+
| DSA S Sign Requests-Succ:                              0 |
| DSA S Sign Request-Err:                                0 |
| DSA S Sign Completed-Succ:                             0 |
```



DSA S Sign Completed-Err:	0	
+-----+		
DSA RS Sign Requests-Succ:	20	
DSA RS Sign Request-Err:	0	
DSA RS Sign Completed-Succ:	20	
DSA RS Sign Completed-Err:	0	
+-----+		
DSA Verify Requests-Succ:	20	
DSA Verify Request-Err:	0	
DSA Verify Completed-Succ:	20	
DSA Verify Completed-Err:	0	
DSA Verify Completed-Failure:	0	
+-----+		
RSA Stats		
+-----+		
RSA Key Gen Requests:	20	
RSA Key Gen Request Errors	0	
RSA Key Gen Completed:	20	
RSA Key Gen Completed Errors:	0	
+-----+		
RSA Encrypt Requests:	0	
RSA Encrypt Request Errors:	0	
RSA Encrypt Completed:	0	
RSA Encrypt Completed Errors:	0	
+-----+		
RSA Decrypt Requests:	20	
RSA Decrypt Request Errors:	0	
RSA Decrypt Completed:	20	
RSA Decrypt Completed Errors:	0	
+-----+		
Diffie Hellman Stats		
+-----+		
DH Phase1 Key Gen Requests:	40	
DH Phase1 Key Gen Request Err:	0	
DH Phase1 Key Gen Completed:	40	
DH Phase1 Key Gen Completed Err:	0	
+-----+		
DH Phase2 Key Gen Requests:	40	
DH Phase2 Key Gen Request Err:	0	
DH Phase2 Key Gen Completed:	40	
DH Phase2 Key Gen Completed Err:	0	
+-----+		
Key Stats		
+-----+		
SSL Key Requests:	0	
SSL Key Request Errors:	0	
SSL Key Completed	0	
SSL Key Complete Errors:	0	
+-----+		
TLS Key Requests:	0	
TLS Key Request Errors:	0	
TLS Key Completed	0	
TLS Key Complete Errors:	0	
+-----+		

4.9 Heartbeat Feature and Recovery from Hardware Errors

The PCH can detect and report to the acceleration driver typically unrecoverable hardware errors that the driver can recover from by resetting and restarting the device. Additionally, the "Heartbeat" feature allows detection and recovery from software/firmware errors in the PCH.



The Acceleration driver can optionally reset the device in the event of an admin message timeout or a heartbeat query failure. The timeout or heartbeat query failure indicates that the firmware running on the Accelerator has become unresponsive. This can happen when an application sends invalid data, for example, invalid source data, or an invalid output data pointer.

Note: Recovery on detection of a Heartbeat failure is not enabled by default. Automatic recovery can be enabled by building the acceleration software with a compile-time flag. The ICP_HEARTBEAT compile-time flag enables this functionality. When the driver is not built with this flag, the acceleration software writes a message to the system (/var/log/messages), reporting that the device is not responding and the device will need to be restarted by the user.

The firmware, if healthy, responds with request/response counters for each accelerator engine on the device. If the firmware is not responsive, a timeout occurs. When such a condition is detected, the driver notifies applications by calling a notification callback for each instance that is registered for notification callback. The event type in this case is CPA_INSTANCE_EVENT_RESTARTING. Then, the device is restarted and all resources allocated to the device, except instance handles, are freed. After restart, all resources are reallocated and the driver notifies applications by calling a notification callback for every instance. The event type in this case is CPA_INSTANCE_EVENT_RESTARTED. Thereafter, the application can use all instances and no further initialization is required. When an application tries to use any instance that uses a restarting device, a new return code CPA_STATUS_RESTARTING is returned. If there is more than one PCH device in the system, and one device is restarted, applications can still use instances on other devices.

4.9.1 How to Call the Heartbeat Query

The Heartbeat query is not kicked off by the driver, it must be initiated by the user. It can be initiated using any of the following methods:

- Watch on `cat /proc/icp../qatx`
- Periodically call heartbeat APIs (see [User Application Heartbeat APIs \(not Enabled by Default\)](#) on page 48).

It will report “QAT is not responding” message in the case that the firmware threads hangs. The device will need to be reset to recover from this error. By default, the device does not automatically reset. It can be manually reset using `adf_ctl <deviceId> reset`.

4.9.1.1 User Proc Entry Read (not Enabled by Default)

The user can periodically perform a read of the /proc entry as specified by any one of the following methods:

Note: The examples below are for one device on one accelerator. The user should apply the desired method to each device and accelerator of interest.

- Manually from command line using the command:

```
# cat /proc/icp_dh89xxcc_dev0/qat0
```



- From a watch process running in background:

```
# watch -n0.1 cat /proc/icp_dh89xxcc_dev0/qat0 > /dev/null
```

- From simple script running in the background:

```
#!/bin/bash
while :
do
    cat /proc/icp_dh89xxcc_dev0/qat0 > /dev/null
    sleep 1
done
```

For example, to send an admin message to device 2, the user issues the following command:

```
# cat /proc/icp_dh89xxcc_dev2/qat0
```

```
+-----+
| Statistics for Qat Instance 0 |
+-----+
| Firmware Requests[AE 0]: 5 |
| Firmware Responses[AE 0]: 5 |
+-----+
| Firmware Requests[AE 1]: 4 |
| Firmware Responses[AE 1]: 4 |
+-----+
```

```
| Firmware Requests[AE 2]: 3 |
| Firmware Responses[AE 2]: 3 |
+-----+
| Firmware Requests[AE 3]: 0 |
| Firmware Responses[AE 3]: 0 |
+-----+
```

If the device is unresponsive and if the acceleration software is built to automatically reset the device on failure, the following message is displayed:

```
ERROR: QAT is not responding and it will be restarted
```

If the device is unresponsive and if the acceleration software is built to not automatically reset the device on failure, the following message is displayed:

```
ERROR: QAT is not responding. Please restart the device
```

4.9.1.2 User Application Heartbeat APIs (not Enabled by Default)

These functions have the following signatures:

```
CpaStatus icp_sal_check_device(Cpa32U accelId);
```

```
CpaStatus icp_sal_check_all_devices(void);
```




See [icp_sal_check_device](#) on page 132 and [icp_sal_check_all_devices](#) on page 133 for details on the functions and parameters.

4.9.2 Handling Heartbeat Failures

The driver must be compiled with ICP_HEARTBEAT defined to do recovery sequence on detecting a heartbeat failure.

A typical heartbeat error use-case is as follows:

1. The driver is loaded, initialized and started.
2. The user-space application registers for instance notifications by calling `cpaCyInstanceSetNotificationCb` and `cpaDcInstanceSetNotificationCb`
3. The application detects that the firmware is unresponsive using the heartbeat feature (see [Heartbeat Feature and Recovery from Hardware Errors](#) on page 46.
4. The kernel-space driver sends the Restarting event to user-space processes.
5. The user-space processes
 - pass the restarting event on to the application instances registered
 - free memory and rings associated with all the instances.
6. The kernel-space driver
 - triggers the device reset (save state, initiate SBR, restore state)
 - once the reset is complete, sends the Restarted event to user-space processes.
7. The user-space processes
 - set up each instance associated with the process, including allocating memory and rings
 - pass the restarted event on to the application instances registered.

Note: If built with ICP_WITHOUT_THREAD then the user-space processes will not automatically get the Restarting and Restarted events. See [Thread-less Mode](#) on page 53.

In a driver built without ICP_HEARTBEAT, there is no automatic recovery on device failure detection. The driver should be reset using `adf_ctl reset` or the `icp_reset_device()` API.

4.9.3 AER and Uncorrectable Errors

Two other errors can be detected that need to be recovered by resetting the device.

- **Uncorrectable errors feature** . Errors detected by the QAT device generate an interrupt handled by the driver. Errors will be seen in the log.
- **Advanced Error Reporting feature** . PCIEAER. If kernel detects an error caused by the driver errors will be seen in the log and the kernel can trigger a device reset.

On detecting either of these errors, the device will be automatically reset by the driver.



4.9.4 Handling Device Failures in a Virtualized Environment

The heartbeat feature in the acceleration software can be used in a virtualized environment. Refer to the *Using Intel® Virtualization Technology (Intel® VT) with Intel® QuickAssist Technology Application Note* for more details on enabling SR-IOV and the creation of Virtual Functions (VFs) from a single Intel® QuickAssist Technology acceleration device to support acceleration for multiple Virtual Machines (VMs).

Note: The Physical Function (PF) driver used here refers to the Intel® QuickAssist Technology PF driver. The Virtual Function (VF) driver used here refers to the Intel® QuickAssist Technology VF driver.

The following sequence describe a possible use case for using the heartbeat feature in a virtualized environment.

1. The PF driver is loaded, initialized and started.
2. The VF driver is loaded, initialized and started in the Guest OS in the VM.
3. The PF driver detects that the firmware is unresponsive (using either of the following methods: [User Proc Entry Read \(not Enabled by Default\)](#) on page 47 or [User Application Heartbeat APIs \(not Enabled by Default\)](#) on page 48).
4. The PF driver sends the "Restarting" event message to the VF via the internal PF-to-VF communication messaging mechanism.
5. The VF driver sends the "Restarting" event to the application's registered callback (the callback is registered using the `cpaDcInstanceSetNotificationCb()` or `cpaCyInstanceSetNotificationCb()` Intel® QuickAssist Technology API function) in the Guest OS.
 - The application's callback function may perform any application-level cleanup.
6. The return from the application's callback triggers the VF driver to send an ACK message back to the PF driver. At this time:
 - The application may perform a complete shutdown.
 - The user may force a graceful shutdown of the Guest OS in the VM.
7. The PF driver receives the ACK message from the VF driver (a timeout mechanism is used to handle any unexpected condition).
8. The PF driver starts the reset sequence (save state, initiate reset, and restore state).
9. The user restarts the Guest OS and loads the VF driver and application in the Guest OS.

Note: If the heartbeat feature in the acceleration software is not enabled, the PF driver will not notify the VF driver that the firmware is unresponsive.

Note: If built with `ICP_WITHOUT_THREAD` then the user-space processes will not automatically get the Restarting and Restarted events. See [Thread-less Mode](#) on page 53.

Note: The error detection mechanisms are not available on the VF driver in the VM, but device errors caused by any of the software running on the VM will be detected by the PF driver using the above mechanisms.



4.9.5 GbE Watchdog Service

The GbE Watchdog Service (`gige_watchdog_service`) is provided to properly reset and restart the GbE interfaces on an Intel® Communications Chipset 8900 to 8920 Series (PCH) device on detecting a reset of the device by the Heartbeat functionality in the Acceleration driver. The user of these GbE interfaces on the PCH device may get an intermittent network disconnect and reconnect as the GbE interfaces are reset and restarted by this service. This service is automatically enabled and started when the Acceleration software is installed with the installation script.

Notes:

- The GbE Watchdog Service is included in the Acceleration software, but it can be considered a separate service. That is to say, it is not integrated into the driver.
- This GbE Watchdog Service does not affect other GbE interfaces available on the system that are not on the PCH device.
- If the GbE interfaces on the PCH device are not used, the GbE Watchdog Service must be disabled and the GbE driver (`igb`) must not be loaded/installed on the system.
- If the GbE interfaces are not used and the GbE driver is loaded/installed when the Heartbeat feature resets the PCH device, the system may become unstable and unresponsive.

4.9.6 Special Considerations When Using the Heartbeat Feature and the GbE Watchdog Service

When using the Heartbeat functionality in the acceleration software with the GbE Watchdog Service, special considerations may need to be taken into account in specific use cases. The following table shows the recommended action(s) when using the Heartbeat feature with/without GbEs on the PCH and with/without external GbEs from Intel in the system.

Table 2. Heartbeat/GbE Watchdog Service Scenarios

Heartbeat Enabled?	GbEs on PCH Enabled	External Intel GbEs Enabled	Recommended Action(s)
No	Yes	No	Disable GbE Watchdog Service
Yes	Yes	No	Enable GbE Watchdog Service
No	No	No	Perform blacklist <code>igb</code> and disable GbE Watchdog Service
Yes	No	No	Perform blacklist <code>igb</code> and disable GbE Watchdog Service
Yes	No	Yes	Either: <ul style="list-style-type: none"> • Turn off all GbEs on the PCH (<code>ifdown</code>) OR • Modify the <code>igb</code> driver to remove the PCI device ID of GbEs on the PCH and recompile the <code>igb</code> driver AND disable the GbE Watchdog Service
Yes	Yes	Yes	Enable GbE Watchdog Service
No	Yes	Yes	Disable GbE Watchdog Service
No	No	Yes	Disable GbE Watchdog Service



Notes:

- "Heartbeat Enabled" with "Yes" means that the acceleration software has the Heartbeat feature enabled (that is, the acceleration software is built with the ICP_HEARTBEAT compile-time flag).
- "Heartbeat Enabled" with "No" means that the acceleration software has the Heartbeat feature disabled (that is, the default case where the acceleration software is built without the ICP_HEARTBEAT compile-time flag).
- "GbEs on PCH Enabled" with "Yes" means that the igb driver for the GbEs on the PCH is loaded/installed and the interfaces are up (ifup). This igb driver may also support other external Intel GbEs.
- "GbEs on PCH Enabled" with "No" means that the igb driver for the GbEs on the PCH is not loaded/installed and the interfaces are down (ifdown). This igb driver may also support other external Intel GbEs.
- "External Intel GbEs Enabled" with "Yes" means that the igb driver for the external Intel GbEs is loaded/installed and the interfaces are up (ifup). This igb driver may also support the GbEs on the PCH.
- "External Intel GbEs Enabled" with "No" means that the igb driver for the external Intel GbEs is not loaded/installed and the interfaces are down (ifdown). This igb driver may also support the GbEs on the PCH.

4.10 Driver Threading Model

By default, when an application uses the acceleration driver in user space, the driver creates threads internally.

When the application calls the `icp_sal_userStart()` or `icp_sal_userStartMultiProcess()` function, the driver creates the following threads:

- **Monitor Thread**

There is only one instance of this thread per system. It loops infinitely and checks if new devices become active in the system that the user proxy layer can start using. If it finds such a device, it spawns a listener thread for that device and continues.

- **Listener Thread**

There is one listener thread per active device in the system. A listener thread calls a blocking read function on the `/dev/icp_dev<N>_csr` file, which blocks until there are device events, such as `EVENT_INIT`, `EVENT_START`, `EVENT_STOP`, `EVENT_SHUTDOWN`, `EVENT_RESTARTING` or `EVENT_RESTARTED` that need to be delivered to user space. If the thread gets an event, it sends it to all user space subsystems (ADF, SAL) and calls the blocking read again in a loop. In the case of a shutdown event, the thread delivers the event and finishes.

- **Ring Thread**

Ring threads are only created for IRQ-driven service instances in user space. If all instances are polled, no ring thread is created. For each IRQ driver response (Rx) ring created in user space, there is one worker thread. User callbacks are called in the context of this worker thread. Additionally, one dispatcher thread (per device) is created when the first Rx ring is allocated (and exits when the last Rx ring is freed). This thread waits for IRQs that are delivered by the kernel space driver and dispatches jobs to worker threads.



4.10.1 Thread-less Mode

The user sets an environment variable:

```
setenv ICP_WITHOUT_THREAD = 1
```

When the driver is built with this flag set, no threads are created by the User Space driver.

In this mode, no IRQ-driven instances are allowed and no events from kernel driver are propagated to user space automatically (with the exception of the first `EVENT_INIT` and `EVENT_START` events).

There are two new API functions that can be used in this mode:

- `CpaStatus icp_sal_find_new_devices(void)` - Performs a function similar to the monitor thread, that is, checks if there are new devices in the system.
- `CpaStatus icp_sal_poll_device_events(void)` - Performs a function similar to the listener thread, that is, polls for events.

It is the user's responsibility to use these functions to monitor the state of devices and receive device-related events.

4.11 Compression Status Codes

The `CpaDcRqResults` structure should be checked for compression status codes in the `CpaDcReqStatus` data field. The mapping of the error codes to the enums is included in the `quickassist/include/dc/cpa_dc.h` file.

4.11.1 Intel® QuickAssist Technology Compression API Errors

The two traditional Intel® QuickAssist Technology Compression APIs, `cpaDcCompressData()` and `cpaDcDecompressData()`, that send requests to the compression hardware can return the error codes shown in the following table.

Table 3. Intel® QuickAssist Technology Compression API Errors

Error Code	Error Type	Description	Suggested Corrective Action(s)
0	CPA_DC_OK	No error detected by compression hardware.	None.
-1	CPA_DC_INVALID_BLOCK_TYPE	Invalid block type (type = 3); invalid input stream detected for decompression; for dynamic compression, corrupted intermediate data	Discard output; resubmit affected request or abort session.
-2	CPA_DC_BAD_STORED_BLOCK_LEN	Stored block length did not match one's complement; invalid input stream detected	Discard output; resubmit affected request or abort session.
-3	CPA_DC_TOO_MANY_CODES	Too many length or distance codes; invalid input stream detected; for dynamic compression, corrupted intermediate data	Discard output; resubmit affected request or abort session.
<i>continued...</i>			



Error Code	Error Type	Description	Suggested Corrective Action(s)
-4	CPA_DC_INCOMPLETE_CODES_LENS	Code length codes incomplete; invalid input stream detected; for dynamic compression, corrupted intermediate data	Discard output; resubmit affected request or abort session.
-5	CPA_DC_REPEATED_LENS	Repeated lengths with no first length; invalid input stream detected; for dynamic compression, corrupted intermediate data	Discard output; resubmit affected request or abort session.
-6	CPA_DC_MORE_REPEAT	Repeat more than specified lengths; invalid input stream detected; for dynamic compression, corrupted intermediate data	Discard output; resubmit affected request or abort session.
-7	CPA_DC_BAD_LITERAL_CODES	Invalid literal/length code lengths; invalid input stream detected; for dynamic compression, corrupted intermediate data	Discard output; resubmit affected request or abort session.
-8	CPA_DC_BAD_DISTANCE_CODES	Invalid distance code lengths; invalid input stream detected; for dynamic compression, corrupted intermediate data	Discard output; resubmit affected request or abort session.
-9	CPA_DC_INVALID_CODE	Invalid literal/length or distance code in fixed or dynamic block; invalid input stream detected; for dynamic compression, corrupted intermediate data	Discard output; resubmit affected request or abort session.
-10	CPA_DC_INVALID_DIST	Distance is too far back in fixed or dynamic block; invalid input stream detected; for dynamic compression, corrupted intermediate data	Discard output; resubmit affected request or abort session.
-11	CPA_DC_OVERFLOW	Overflow detected. This is not an error, but an exception. Overflow is supported and can be handled.	Continue with the session as normal.
-12	CPA_DC_SOFTERR	Other non-fatal detected.	Discard output; resubmit affected request or abort session.
-13	CPA_DC_FATALERR	Fatal error detected.	Discard output; restart or reset session.

Except for the errors, `CPA_DC_OK`, `CPA_DC_OVERFLOW`, and `CPA_DC_FATALERR`, the rest of the error codes can be considered as invalid input stream errors.

4.12 Stateful Compression - Dealing with Error Code `CPA_DC_BAD_LITERAL_CODES (-7)`

Prior to software release version 1.2, the driver was unable to deal with the `CPA_DC_BAD_LITERAL_CODES (-7)` error code being returned from the acceleration software. A software workaround has been implemented to overcome this hardware deficiency.

Error -7 occurs when running a stateful decompression. Stateful decompression uses some history that is stored in the internal memory of the data compression hardware.



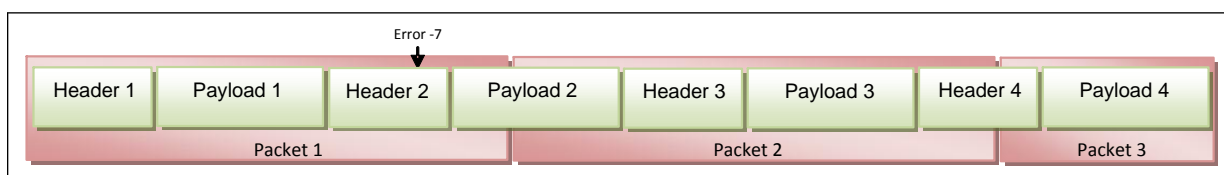
For some hardware specific reasons, this internal memory is corrupted when the acceleration software tries to deal with certain packet headers. The header is incorrectly decoded, computed data in the internal memory is incorrect, and the error (-7) is generated.

To overcome this issue, a workaround has been implemented that searches for the faulty header in the source data packet that the acceleration software is trying to decompress. When the header is found, the acceleration software computes and loads the data that should have been in the internal memory. With this internal memory loaded with the correct data, a call to the `cpaDcDecompressData()` function is required to finish decompressing the rest of the packet.

4.12.1 Example of a Stream that Triggers Error Code (-7)

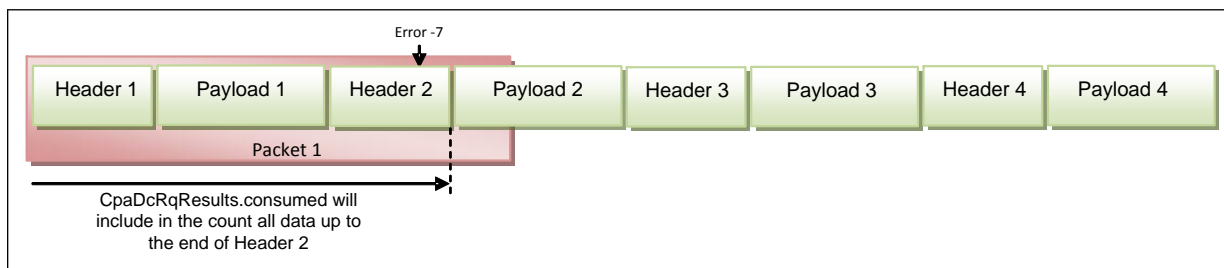
The following figure shows an example of a stream comprising four headers and their corresponding payloads. Let us assume that Header 2 produces error code (-7) and that the user cuts the stream in the middle of Payload 2 and Payload 4 to form three packets.

Figure 15. Stream of Compressed Data Split into Three Packets



As a result of the software workaround included in Release 1.2 and later, the data is processed up to the beginning of Header 2. When the acceleration software encounters Header 2, "error code (-7)" is hidden from the user and the software workaround processes the header and updates data in internal hardware memory.

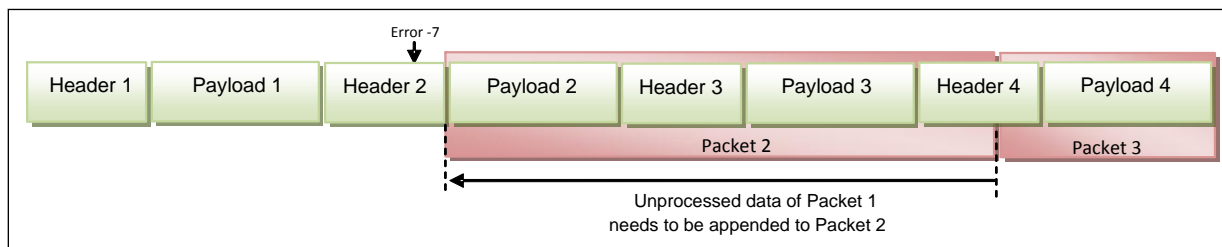
Figure 16. Accelerator Software Consumes Data up to End of Header 2 where Error Code (-7) is Normally Generated



However, the software workaround is unable to decompress the rest of the packet, and therefore it is the user's responsibility to call the `cpaDcDecompressData()` function again on the remaining data.

The most efficient approach is to check the `CpaDcRqResults.consumed` field returned by the `cpaDcDecompressData()` function and see if all the data in the source buffer have been consumed. If not, Intel recommends including the unprocessed data in the next packet as shown in the following figure.

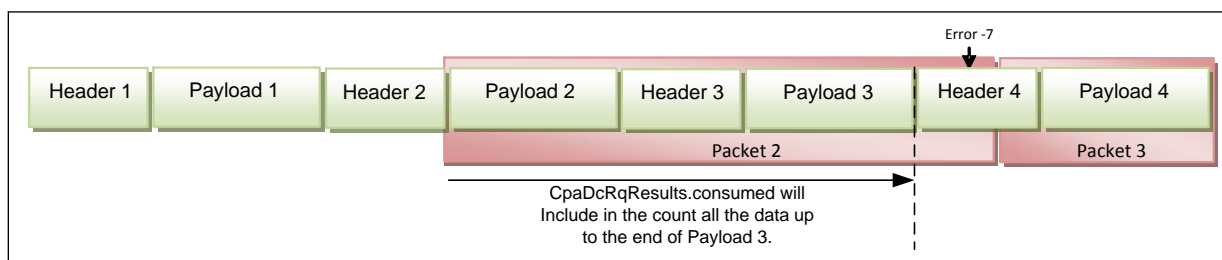
Figure 17. Unprocessed Data Appended to Next Packet



4.12.2 Special Case when a Packet Cuts a Header in the Stream

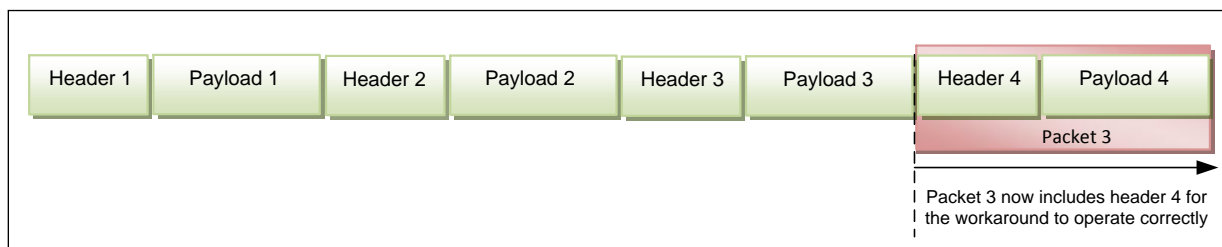
The following figure shows a packet cut in the middle of a header that triggers error code (-7). The `cpaDcDecompressData()` function returns no error, but as in the example described in [Example of a Stream that Triggers Error Code \(-7\)](#) on page 55, not all the packet data is consumed. Therefore, the user must compare the consumed data with the original packet size.

Figure 18. Packet Cut in the Middle of a Header



By doing so, the user will determine that the truncated header has not been consumed and consumed data includes up to the end of payload 3. Next, the user must prepend the unprocessed data of packet 2 to packet 3 and submit the request. By doing so, this includes all the data necessary for the workaround to operate correctly. The following figure shows the new format of packet 3.

Figure 19. New Format of Packet 3 with Data Prepended from Previous Requests



4.12.3 Pseudo Code for Handling Error Code -7

The following pseudo code shows how to handle error code -7 generated during stateful decompression.

```
BEGIN
  Buffer_offset = 0
DO
```




```

    Read stream and store data starting from Buffer_offset

    Packet size = 0
    For all the buffers in SGL source buffer list
        Packet size = Packet size + current buffer data length in bytes
    Next buffer

    Remaining bytes to decompress = Packet size

    Call cpaDcDecompressData() API function

    IF CpaDcRqResults.status ≠ OK THEN
        Return Error
    ENDIF

    Remaining bytes to decompress = Remaining bytes to decompress -
    CpaDcRqResults.consumed

    Buffer_offset = Remaining bytes to decompress

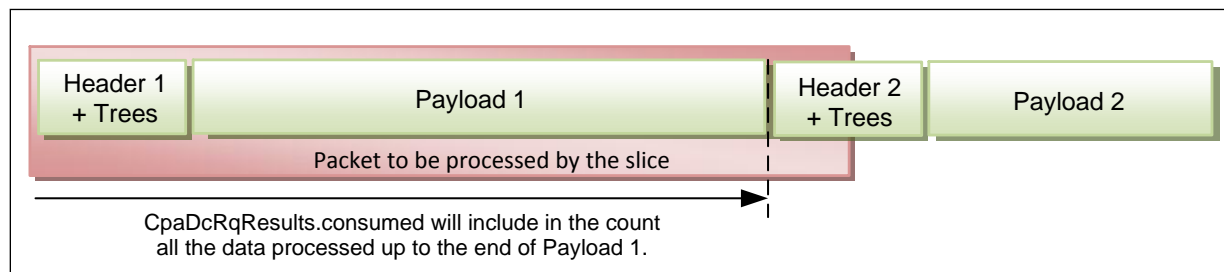
    IF Remaining bytes to decompress > 0 THEN
        Find buffer index and buffer offset in SGL of last consumed data byte
        Prepend unprocessed data (From last process data to the end of the
        last buffer) to SGL source buffer list.
    ENDIF
    LOOP until end of stream
END

```

4.12.4 Unprocessed Data During Stateful Decompression Operations

When running stateful decompression operations, the user may observe in some cases that not all of the data is consumed by the slice, but the `cpaDcDecompressData()` API returns `CPA_STATUS_SUCCESS`. This can occur in two cases:

- A packet with an odd number of bytes: The slice was designed to operate on packets size with multiples of 2 bytes. At the API level, the user is free to allocate the buffer size that they want, but if the user submits 17 bytes to be inflated, the `cpaDcDecompressData()` API reports 16 bytes consumed. The user must then take the unprocessed byte and prepend it to the next packet. If the user omits this step, the compression history will be broken and the slice returns an error on the next request.
- A packet contains an incomplete header: This use case occurs when running dynamic stateful decompression. If the packet to be processed has an incomplete header, the slice cannot process the Huffman trees. In this case, the slice reports consumed data up to the beginning of the incomplete header. The following figure shows the use case. Header 2 is incomplete and the slice consumes data up to the beginning of Header 2.

Figure 20. Unprocessed Data for Incomplete Header


When doing stateful decompression, the user must always check the number of bytes consumed even if the `status` parameter of the `CpaDcRqResults` structure returns `CPA_STATUS_SUCCESS`.

4.13 Stateful Compression Level Details

Throughput and compression ratio for stateful compression can be adjusted with the compression levels to achieve particular requirements. The following table shows the mapping of the compression levels to the history window, search depth, and context size.

Note: The State registers are also saved.

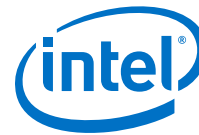
Compression Level	History Windows	Search Depth	Context Size
1	32 kB	1	48 kB
2	8 kB	4	48 kB
3	8 kB	8	48 kB
4-9	8 kB	16	48 kB

4.14 Stateless Compression Level Details

Throughput and compression ratio for stateless compression can be adjusted with the compression levels to achieve particular requirements. The following table shows the mapping of the compression levels to the history window, search depth, and context size.

Note: No context is saved and no State registers are saved.

Compression Level	History Windows	Search Depth	Context Size (Kbyte)
1	32 kB	1	0
2	8 kB	4	0
3	8 kB	4	0
4-9	8 kB	16	0



4.15 Acceleration Driver Error Scenarios

This section describes the behavior of the Acceleration Driver in various error scenarios.

4.15.1 User Space Process Crash

Error Scenario	A user space process crashes without cleanly stopping the user space acceleration driver in the process.
Background	<p>The kernel acceleration driver keeps track of all rings created by each process on a device. From the user space acceleration driver, rings are created on a device via <code>ioctl</code> calls on <code>icp_dev<N>_ring</code>. The kernel acceleration driver maintains a list of rings per pid, per device.</p> <p>In a similar way, the kernel acceleration driver keeps track of all internal memory allocation. Physically contiguous memory chunks are allocated from the user space acceleration driver via <code>ioctl</code> calls on <code>icp_dev_mem</code>. The kernel driver keeps track of all memory allocated per pid.</p> <p>These files are opened at initialization when an application calls <code>icp_sal_userStart()</code> and are closed when an application calls <code>icp_sal_userStop()</code> or closed by the operating system when the application is killed/crashed.</p>
Sequence of Events	<ol style="list-style-type: none"> 1. The user space process crashes. 2. The OS calls a release handler in the kernel acceleration driver, with the pid of the crashed process, for each opened <code>/dev/icp_dev_*</code> file. 3. The kernel acceleration driver frees any allocated resources (rings/memory) associated with the crashed process. <ol style="list-style-type: none"> a. For memory allocations, the kernel acceleration driver frees all the memory buffers in the list. b. For rings, the kernel acceleration driver creates a new list and starts an "orphan" thread (if it is not running at the given time) and passes the list of rings associated with the process to the orphan thread. The orphan thread then loops and waits for all the in-flight requests to come back, then it frees the rings.
Side Effects	None

4.15.2 Hardware Hang Detected by Heartbeat

Error Scenario	Acceleration hardware hangs, for example, due to a bad DMA address passed to the driver and hardware. A device reset is required to recover from the hang. The hang is detected by a "heartbeat" poll that triggers a reset of the acceleration device. The reset happens if and only if the Heartbeat feature is enabled using the compile-time option.
Sequence of Events	<ol style="list-style-type: none"> 1. Applications register for instance notifications by calling <code>cpaCyInstanceSetNotificationCb()</code> and <code>cpaDcInstanceSetNotificationCb()</code>. 2. Applications must periodically issue a "heartbeat" poll via either an API call to either <code>icp_sal_check_device()</code> or <code>icp_sal_check_all_devices()</code> or by reading a file in the <code>/proc</code> file system. 3. For each heartbeat poll, the kernel acceleration driver sends SYNC/GET messages to the acceleration hardware and waits for responses. If the driver times out waiting for responses, the driver triggers a reset of the acceleration device. 4. Before resetting the device, the kernel acceleration driver notifies the user space acceleration drivers that the device is about to be reset. 5. Once notified that a device is about to be reset, the user space acceleration driver:

continued...



	<ul style="list-style-type: none">• Sends a CPA_INSTANCE_EVENT_RESTARTING event to registered applications.• Frees memory and rings associated with all the instances. <ol style="list-style-type: none">6. After the reset is complete, the kernel acceleration driver notifies the user space acceleration driver that the reset is complete.7. Once notified that a device reset is complete:<ul style="list-style-type: none">• Set up each instance associated with the process. This includes allocating memory and rings for each instance.• Send a CPA_INSTANCE_EVENT_RESTARTED event to registered applications.
Side Effects	On a device reset, the PCH Gigabit Ethernet devices are also reset. The GigE drivers can recover from this reset by running a GigE watchdog process. This watchdog will be notified by the acceleration driver before a reset and the watchdog will shut down the network interfaces of each effected GigE. The kernel space acceleration driver then saves the state of each GigE. Following the reset, the kernel acceleration driver restores the GigE state and notifies the watchdog process which then brings the network interfaces back up. See Heartbeat Feature and Recovery from Hardware Errors on page 46 for further details.

4.15.3 Hardware Error Detected by AER

Error Scenario	Acceleration hardware detects an un-correctable error. A device reset is needed to recover from the error.
Sequence of Events	<ol style="list-style-type: none">1. Acceleration hardware detects an un-correctable error. It notifies the kernel acceleration driver via an error interrupt.2. If, and only if the Heartbeat feature is enabled by the ICP_HEARTBEAT compile-time option, the kernel acceleration driver resets the acceleration device upon receipt of the interrupt. The reset sequence follows the same flow as steps 4 to 7 in Hardware Hang Detected by Heartbeat on page 59.
Side Effects	Same as Hardware Hang Detected by Heartbeat on page 59.

4.15.4 Virtualization: User Space Process Crash (in Guest OS)

Error Scenario	A user space process running in a guest OS within a Virtual Machine (VM) crashes. It is assumed that the user space process is using an Intel® QuickAssist Technology Virtual Function (VF) that has been assigned to the VM.
Sequence of Events	Within the VM, the sequence of events is the same as for the non-virtualization error scenario, see User Space Process Crash on page 59. There is no involvement from the Intel® QuickAssist Technology Physical Function (PF) driver in this scenario.
Side Effects	None

4.15.5 Virtualization: Guest OS Kernel Crash

Error Scenario	A Virtual Machine (VM) crashes in an uncontrolled manner, for example, due to a kernel crash within the guest OS running in the VM.
Background	In a controlled VM shutdown, the Intel® QuickAssist Technology Virtual Function (VF) driver running in the VM the VF from the shutdown VM. The Intel® QuickAssist Technology PF driver keeps track of the ring resources used by each VF.
Sequence of Events	<ol style="list-style-type: none">1. The VM crashes.

continued...



	2. The host OS/VMM detects the VM crash and un-assigns the VF from the crashed VM.
Side Effects	It is possible that there are in-flight requests within the acceleration hardware when the VM crashes. In this scenario, it is possible that memory accesses from the acceleration hardware to the VM memory address space may cause a hardware hang if that address space has been removed from the iommu.

4.15.6 Virtualization: Hardware Hang Detected by Heartbeat

Error Scenario	The acceleration hardware hangs as a result of processing a bad request issued from a Virtual Machine (VM), for example, due to a bad address passed to the acceleration hardware. A full device reset is required to recover from the error.
Sequence of Events	<ol style="list-style-type: none"> 1. The acceleration hardware hang is detected via the heartbeat mechanism running in the host OS/VMM with the Intel® QuickAssist Technology Physical Function (PF) driver. 2. The sequence of events within the host OS is the same as for the non-virtualization scenario. See Hardware Hang Detected by Heartbeat on page 59. 3. Each VF acceleration driver is informed that the device is restarting, and so starts its reset sequence. This will result in the same events being notified to services on the VMs as on the Host.
Side Effects	All VMs that are assigned VFs from the same silicon device are affected.

4.15.7 Virtualization: Hardware Hang Detected by AER

Error Scenario	The acceleration hardware detects an un-correctable error. A device reset is needed to recover from the error.
Sequence of Events	<ol style="list-style-type: none"> 1. The reset sequence is the same as for the non-virtualization scenario. See Hardware Error Detected by AER on page 60. 2. Each VF acceleration driver is informed that the device is restarting, and so starts its reset sequence. This will result in the same events being notified to services on the VMs as on the Host.
Side Effects	All VMs that are assigned VFs from the same silicon device are affected.

4.16 Build Flag Summary

The following tables summarize the options available when building the software.

The following table shows the build flags that must be specified.

Table 4. Required Build Flags

Symbol	Description	Default	Reference
ICP_ROOT	Set to the directory where acceleration software is extracted. This may be /QAT or /QAT/QAT1.5, depending on how the driver was compiled.	User defined	
ICP_BUILDSYSTEM_PATH	Set to the build system folder located under the quickassist folder (\$ICP_ROOT/quickassist/build_system)	User defined	
<i>continued...</i>			



Symbol	Description	Default	Reference
ICP_BUILD_OUTPUT	Set to directory that executable/libraries are placed in (\$ICP_ROOT/build)	User defined	
ICP_ENV_DIR	Set to the directory that contains the environmental build files (\$ICP_ROOT/quickassist/build_system/build_files/env_files)	User defined	
ICP_TOOLS_TARGET	Set to accelcomp for DH89xxcc platforms	User defined	

The following table shows the build flags that can be optionally specified.

Table 5. Optional Build Flags

Symbol	Description	Default	Reference
DISABLE_PARAM_CHECK	When defined, parameter checking in the top-level APIs is performed. This can be set to optimize performance.	Not defined	
DISABLE_STATS	When defined, disables statistics. Disabling statistics can improve performance.	Not defined, therefore statistics are enabled.	
DRBG_POLL_AND_WAIT	When defined, modifies the behavior of <code>cpaCyDrbgSessionInit</code> and the DRBG HT functions to poll for responses internally rather than depending on an external polling thread.	Enabled	DRBG Health Test and cpaCyDrbgSessionInit Implementation Detail on page 128
ICP_LOG_SYSLOG	When defined, enables debug messages to be output to the system log file instead of standard out for user space applications.	Not defined	
ICP_WITHOUT_THREAD	When defined, no user space threads are created when a user space application invokes <code>icp_sal_userStart</code> or <code>icp_sal_userStartMultiProcess</code> .	Not defined	Thread-less Mode on page 53
ICP_HEARTBEAT	When defined, enables automatic device reset on failures detected by the heartbeat mechanism.		Heartbeat Feature and Recovery from Hardware Errors on page 46
ICP_NONBLOCKING_PARTIALS_PERFORM	When defined, results in partial operations not being blocked.	Not defined	Defined when compiling the driver using the <code>installer.sh</code> installation script.
ICP_SRIOV	Indicates whether SRIOV should be enabled in the driver.	Not defined	
ICP_TRACE	Used to enable tracing capability for debug purposes. Once the acceleration driver is compiled with this option, all the Cryptography	Not defined	

continued...



Symbol	Description	Default	Reference
	and Data Compression APIs will expose their parameters to the console for user space applications OR to <code>/var/log/</code> messages in kernel space.		
LAC_HW_PRECOMPUTES	When defined, enables hardware for HMAC precomputes.	Not defined, therefore the driver uses software (dependency on OpenSSL and Linux Crypto API).	See limitations below table.
max_mr	Used to set the number of Miller Rabin rounds for prime operations. Setting this to a smaller value reduces the memory usage required by the driver.	50	
WITH_CPA_MUX	When defined, the driver will be built for the mux environment, i.e., <code>cpa_mux.ko</code> will be built and will expose the Intel® QuickAssist Technology API. The drivers will not export symbols but will instead register with the <code>cpa_mux</code> .	Depends on devices found on the platform. Not defined if devices found can be supported by a single driver. Defined otherwise, e.g., if both DH89xxcc and DH895xcc devices are found.	
ICP_NUM_PAGES_PER_ALLOC	If defined, the memory driver will allocate a 128K memory to be the memory Slab; otherwise it will allocate 2M memory. For kernel versions older than 2.6.32, this variable should be set.	Not defined	See Compiling Acceleration Software on Older Kernels on page 65
ICP_DISABLE_INLINE	When defined, function inlining for functions that cannot be inlined by the compiler is removed to enable compilation of the driver for kernels build without <code>CONFIG_ARCH_SUPPORTS_OPTIMIZED_INLINING</code>	Not defined	

Notes:

The limitations of pre-computes are as follows:

- Hardware pre-computes are not supported with the Data Plane API in kernel space for both HMAC and AES-ECB pre-computes.
- Hardware pre-computes are not supported with the “traditional” API when using polled instances for kernel space.
- For kernel versions 2.6.18 or less, neither hardware not software pre-computes can be used in polled mode or with the Data Plane API, so the driver cannot support any HMAC (qathashmode 1) or GCM/CCM operation with the Data Plane API with kernel mode.

4.17 Running Applications as Non-Root User

This section describes the steps required to run Intel® QuickAssist Technology user-space applications as non-root user. This section uses the user space performance sample code as an example.



Assumptions:

- Intel® QuickAssist Technology software is installed and running
- User space Acceleration Sample code (cpa_sample_code) compiled and the directory has read/write/execution permission for all the users
- Kernel space memory driver (qaeMemDrv.ko) compiled and installed

The following steps should be executed by users with root privilege or root user.

1. Export environmental variables.

```
# export ICP_ROOT=/QAT
```

2. Create a linux group to provide access for all users in that group.

```
# groupadd <group_name>
```

3. Add users to the new group. The group should only have users who need access to the application.

```
# usermod -G <group_name> <user_name_to_add>
```

4. Change group ownership of the following files. By default, the group ownership will be root.

- /dev/icp_dev_processes
- /dev/icp_dev<N>_ring
- /dev/icp_dev<N>_csr
- /dev/icp_adf_ctl
- /dev/icp_dev_mem
- /dev/icp_dev_mem_page

```
# cd /dev/  
# chgrp <group_name> icp_dev_processes icp_dev*_ring icp_dev*_csr  
icp_dev_mem_page icp_dev_mem icp_adf_ctl  
# chmod 660 icp_dev_processes icp_dev*_ring icp_dev*_csr  
icp_dev_mem_page icp_dev_mem icp_adf_ctl
```

5. Change the File permission for the following configuration files to 644.

```
# chmod 644 /etc/dh89?xcc_qa_dev?.conf
```

6. Change the group ownership for the Intel® QuickAssist Technology user space driver (libicp_qa_al_s.so).

For 64-bit OS:

```
# cd /lib64  
# chgrp <group_name> libicp_qa_al_s.so
```




For 32-bit OS:

```
# cd /lib
# chgrp <group_name> libicp_qa_al_s.so
```

7. Change the group ownership for memory driver.

```
# cd /dev
# chgrp <group_name> qae_mem
# chmod 660 qae_mem
```

8. At this point, switch to user name that is included in <group_name>

```
# su <user name that is included in group_name>
```

9. Launch the performance sample code.

```
# cd $ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/build/
# ./cpa_sample_code signOfLife=1
```

Note: If the user does not have access to the directory, modify group ownership of the ICP_ROOT directory.

```
# chgrp -R <group_name> $ICP_ROOT
```

Or copy the sample code application to a directory can be accessed by the user.

```
# cp $ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/build/
cpa_sample_code /home/tester
```

The same basic steps can be followed to enable non-root access for customer applications accessing the acceleration software. Every time the acceleration software is restarted, step 4 must be completed. Every time the memory driver is started, step 7 must be completed.

4.18 Compiling Acceleration Software on Older Kernels

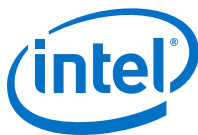
With the current release of the Acceleration software, changes have been added to provide limited support for older kernel versions. These changes allow the driver to compile on kernels as old as the 2.6.18 kernel. They were added to assist customers who are using older kernel versions.

This section describes the steps required in order to compile the acceleration software and describes the limitations of the implementation.

- **Installing**

Define the following environmental variables before compiling the driver. If using the installer.sh script, these can be added to the SetENV() function. If compiling the driver manually, define these variables along with ICP_ROOT, ICP_ENV_DIR, etc.

```
— LAC_HW_PRECOMPUTES=1
— ICP_NUM_PAGES_PER_ALLOC=1
```



Once these are defined, compile and install the driver.

- **Testing**

Once the driver is installed, performance sample code signOfLife tests can be executed. Please refer to the *Intel® Communications Chipset 8900 to 8920 Series Software for Linux* Getting Started Guide* for details.

- **Limitations**

- Older kernels do not support kmallocc of more than 128K. Due to this limitation, compression tests within the performance sample code may not execute.
- Running the performance sample code without the `signOfLife=1` option may fail.
- Ensure `LAC_HW_PRECOMPUTES` is defined if your application uses algorithm chaining from kernel space. The acceleration driver by default makes use of software based hashing for algorithm chaining and this functionality was not available in older kernels. Setting the `LAC_HW_PRECOMPUTES` allows the driver to use hardware acceleration.

4.19 Compiling with Debug Symbols

To compile the driver with debug symbols (for easier debug or for performance profiling), build/rebuild the driver after making the following changes:

1. In `$ICP_ROOT/quickassist/build_system/build_files/OS/linux_2.6.mk`, add the `-g` flag to the user space `CFLAGS`, as shown:

```
ifeq ($(PROG_ACY)_OS_LEVEL, user_space)
CFLAGS+=-fPIC $(DEBUGFLAGS) -g -Wall -Wpointer-arith $(INCLUDES)
```

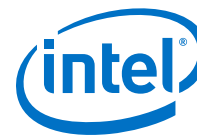
2. In `$ICP_ROOT/quickassist/build_system/build_files/common.mk`, set the optimization level to 0, as shown:

```
#Set default optimization level
$(PROG_ACY)_OPT_LEVEL?=0
EXTRA_CFLAGS+=-O$( $(PROG_ACY)_OPT_LEVEL)
```

4.20 Acceleration Driver Return Codes

The following table shows the return codes used by various components of the acceleration driver.

Return Type	Return Code	Description
CPA_STATUS_SUCCESS	0	Requested operation was successful.
CPA_STATUS_FAIL	-1	General or unspecified error occurred. Refer to the console log user space application or to <code>/var/log/messages</code> in kernel space for more details of the failure.
continued...		



Return Type	Return Code	Description
CPA_STATUS_RETRY	-2	Recoverable error occurred. Refer to relevant sections of the API for specifics on what the suggested course of action.
CPA_STATUS_RESOURCE	-3	Required resource is unavailable. The resource that has been requested is unavailable. Refer to relevant sections of the API for specifics on what the suggested course of action.
CPA_STATUS_INVALID_PARAM	-4	Invalid parameter has been passed in.
CPA_STATUS_FATAL	-5	Fatal error has occurred. A serious error has occurred. Recommended course of action is to shutdown and restart the component.
CPA_STATUS_UNSUPPORTED	-6	The function is not supported, at least not with the specific parameters supplied. This may be because a particular capability is not supported by the current implementation.
CPA_STATUS_RESTARTING	-7	The API implementation is restarting. This may be reported if, for example, a hardware implementation is undergoing a reset.

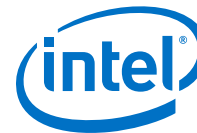
The following table shows the return codes used by the driver to handle Linux device driver operations.

Return Type	Return Code	Description
SUCCESS	0	Operation was successful.
FAIL	1	General error occurred. Refer to the console log user space application or to /var/log/ messages in kernel space for more details of the failure.
-EPERM	-1	Operation is not permitted. Used during ioctl operations.
-EIO	-5	Input/Output error occurred. Used when copying configuration data to and from user space.
-EBADF	-9	Bad File Number. Used when an invalid file descriptor is detected.
-EAGAIN	-11	Try Again. Used when a recoverable operation occurred.
-ENOMEM	-12	Out of Memory. Memory resource that has been requested is not available.
-EACCES	-13	Permission Denied. Used when the operation failed to connect to a process or open a device.

continued...



Return Type	Return Code	Description
-EFAULT	-14	Bad Address. Used when an operation detects invalid parameter data.
-ENODEV	-19	No Such Device. Used when an operation detects invalid device id.
-ENOTTY	-25	Invalid Command Type. Used when an ioctl operation detects an invalid command type.



5.0 Acceleration Driver Configuration File

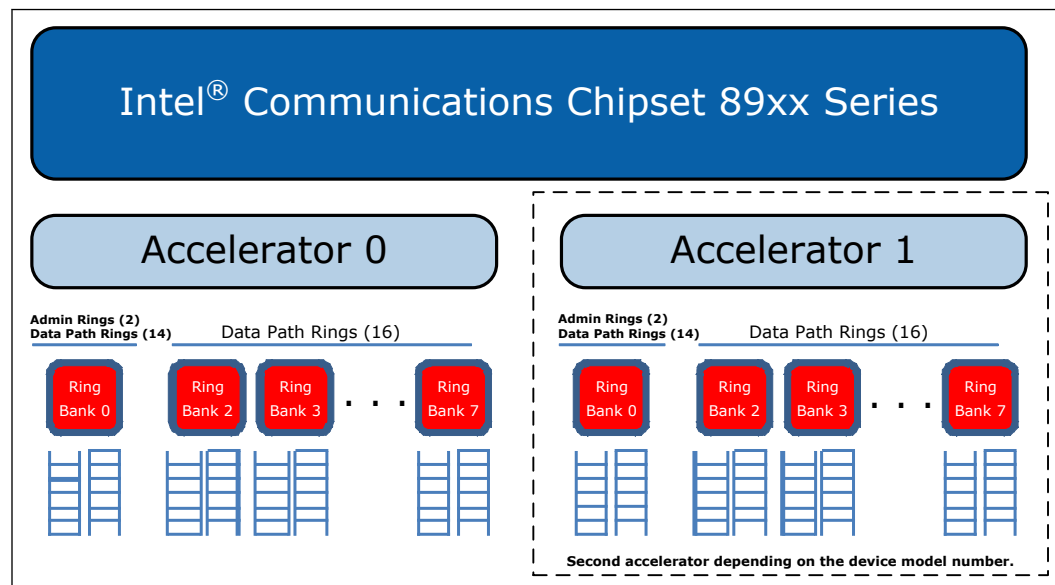
This chapter describes the configuration file(s) managed by the Acceleration Driver Framework (ADF) that allow customization of runtime operation. This configuration file(s) must be tuned to meet the performance needs of the target application.

Note: The software package includes a default configuration file against which optimal performance has been validated. Consider performance implications as well as the configuration details provided in this section if your system requires modifications to the default configuration file.

5.1 Configuration File Overview

There is a single configuration file for each Intel® Communications Chipset 8900 to 8920 Series (PCH) device. A client application can load balance between two accelerators if present. Each accelerator has eight independent ring banks - the communication mechanism between the Acceleration software and the hardware. Each ring bank has an interrupt that can be directed to a specific Intel® architecture core. Each ring bank has 16 rings (hardware assisted queues). This hierarchy is shown in the following figure.

Figure 21. Ring Banks



Note: Depending on the model number, a PCH device may also contain no accelerators.

The configuration file is split into a number of different sections: a General section and one or more Logical Instance sections.

- **General** - includes parameters that allow the user to specify:



- Which services are enabled.
- The configuration file format.
- Firmware location configuration.
- Concurrent request default configuration.
- Interrupt coalescing configuration (optional).
- Statistics gathering configuration.

Additional details are included in [General Section](#) on page 70.

Note: The concurrent request parameters include both transmit (Tx) and receive (Rx) requests. For example, if a concurrent request parameter is set to 64, this implies 32 requests for Tx and 32 for Rx.

- **Logical Instances** - one or more sections that include parameters that allow the user to set:
 - The number of cryptography or data compression instances being managed.
 - For each instance, the name of the instance, the accelerator number, whether polling is enabled or not and the core to which an instance is affinitized.

Additional details are included in [Logical Instances Section](#) on page 74.

A sample configuration file, targeted at a high-end IPsec box, is included in [Sample Configuration File \(V2\)](#) on page 84.

5.2 General Section

The general section of the configuration file contains general parameters and statistics parameters.

5.2.1 General Parameters

The following table describes the parameters that can be included in the General section:

Table 6. General Parameters

Parameter	Description	Default	Range
ConfigVersion	Used to signify the simpler configuration file format. If this parameter is present, the configuration file is in a new format that requires fewer parameter definitions. If this parameter is not present, this implies this is V1 configuration file. V1 configuration files are 100% compatible with this software release.	2	Integer
ServicesEnabled	Defines the service(s) available (cryptographic [cyX], data compression [dc]).	cy0;cy1;dc	cyX, dc

continued...



Parameter	Description	Default	Range
			<p><i>Note:</i> X can be 0 or 1, which identifies one of two available cryptographic engines.</p> <p><i>Note:</i> Multiple values permitted, use ; as the delimiter.</p>
cyHmacAuthMode	Determines when HMAC precomputes are done.	1	<p>1 - HMAC precomputes are done during session initialization</p> <p>2 - HMAC precomputes are done during the perform operation</p> <p><i>Note:</i> In general, with this parameter set to 1, performance is expected to be better.</p>
dcTotalSRAMAvailable	Each PCH device has a total of 512 KB of eSRAM. The eSRAM can be used by the Data Compression service only. This parameter tells the driver how much of this memory to use for the Data Compression service. A value of 0 means, do not use any eSRAM for the Data Compression service; 524288 means use all the eSRAM for the Data Compression service. If an odd value is specified, internally the driver rounds the value down to the nearest even value, for example, if a value of 262145 is specified, the driver rounds the value down to 262144.	0	0 to 524288
Firmware_MofPath	Name of the Microcode Object File (MOF) firmware.	mof_firmware.bin	mof_firmware.bin
Firmware_MmpPath	Name of the Modular Math Processor (MMP) firmware.	mmp_firmware.bin	mmp_firmware.bin
CyNumConcurrentSymRequests	Specifies the number of cryptographic concurrent symmetric requests for cryptographic instances in general. <i>Note:</i> This parameter value can be overridden for a particular cryptographic instance if necessary.	512	64, 128, 256, 512, 1024, 2048 or 4096
CyNumConcurrentAsymRequests	Specifies the number of cryptographic concurrent asymmetric requests for cryptographic instances in general.	64	64, 128, 256, 512, 1024, 2048 or 4096
continued...			



Parameter	Description	Default	Range
	Note: This parameter value can be overridden for a particular cryptographic instance if necessary.		
DcNumConcurrentRequests	Specifies the number of data compression concurrent requests for data compression instances in general. Note: This parameter value can be overridden for a particular data compression instance if necessary.	512	64, 128, 256, 512, 1024, 2048 or 4096
InterruptCoalescingEnabled Note: This parameter is optional.	Specifies if interrupt coalescing is enabled for ring banks.	1	0 or 1
InterruptCoalescingTimerNs Note: This parameter is optional.	Specifies the coalescing time, in nanoseconds (ns) for ring banks. Note: If a value outside the range is set, the default value is used.	10000	500 to 1048575
InterruptCoalescingNumResponses Note: This parameter is optional.	Specifies the number of responses that need to arrive from hardware before the interrupt is triggered. It can be used to maximize throughput or adjust throughput latency ratio.	0 (disable)	0 to 248
ProcDebug	Debug feature. When set to 1 enables additional entries in the / proc file system.	0 (disable)	0 or 1
drbgPollAndWaitTimeMS	An optional parameter that specifies the polling interval (in milliseconds) used when DRBG_POLL_AND_WAIT is defined. Refer to DRBG Health Test and cpaCyDrbgSessionInit Implementation Detail on page 128.	10	1 to 20
SRIOV_Enabled	Enables or disables Single Root Complex I/O Virtualization. If enabled (set to 1), SRIOV and VT-d must be enabled in the BIOS. If disabled (set to 0), then SRIOV and VT-d must be disabled in the BIOS.	0 (disabled)	0 or 1
PF_bundle_offset	When using virtualization and the version 2 configuration file, this parameter indicates the first bank on which to allocate instances for the Physical Function (PF). For example, when PF_bundle_offset = 5, instances in the PF are allocated starting from bank 5, therefore the first five banks (0 to 4) per PCH device are free and available to be assigned to Virtual Machines (VMs). Note: This param should be commented out in the .conf file if the PF will not use any instances. Note: This parameter should not be used if the version 1 configuration file is used.	1	1 to 7
continued...			



Parameter	Description	Default	Range
	Note: Banks 0 and 8 are used for administration messages and therefore cannot be used for services, either PF or VF.		
<p>Note: "Default" denotes the value in the configuration file when shipped.</p> <p>Note: The concurrent request parameters include both transmit (Tx) and receive (Rx) requests. For example, if a concurrent request parameter is set to 64, this implies 32 requests for Tx and 32 for Rx.</p>			

5.2.2 Statistics Parameters

The following table shows the parameters in the configuration file, prefixed with stats, that can be used to enable or disable certain types of statistics.

Note: There is a performance impact when statistics are enabled. In particular, the IA cost of offload is expected to increase when statistics are enabled.

When the statistics are enabled, the collected data can be retrieved using the following methods:

- Calling the appropriate Intel® QuickAssist Technology API function. For example, `cpaCySymQueryStats` or `cpaCySymQueryStats64` for symmetric cryptography. See the *Intel® QuickAssist Technology Cryptographic API Reference Manual* for more information about these functions.
- For kernel space instances, looking at entries in the `/proc/dh89xxcc_devX` directory, where X is the device number. For example, `/proc/icp_dh89xxcc_dev0/cy/IPSec0` for all statistics related to cryptography instance IPSec0, where IPSec0 is the name given to the instance in the config file (Cy0Name = "IPSec0"). See [Debug Feature](#) on page 44 for more information.

Table 7. Statistics Parameters

Parameter	Description	Default	Range
statsGeneral	Enables/disables statistics in general.	1	1 or 0
statsDc	Enables/disables statistics for data compression.	1	1 or 0
statsDh	Enables/disables statistics for the Diffie-Hellman algorithm.	1	1 or 0
statsDrbg	Enables/disables statistics for the Deterministic Random Bit Generator (DRBG).	1	1 or 0
statsDsa	Enables/disables statistics for the Digital Signature Algorithm (DSA).	1	1 or 0
statsEcc	Enables/disables statistics for Elliptic Curve Cryptography (ECC).	1	1 or 0
statsKeyGen	Enables/disables statistics for the Key Generation algorithm.	1	1 or 0
statsLn	Enables/disables statistics for the Large Number generator.	1	1 or 0
statsPrime	Enables/disables statistics for the Prime Number detector.	1	1 or 0
continued...			



Parameter	Description	Default	Range
statsRsa	Enables/disables statistics for the RSA algorithm.	1	1 or 0
statsSym	Enables/disables statistics for symmetric ciphers.	1	1 or 0
<i>Note:</i> "Default" denotes the value in the configuration file when shipped. A value of 1 indicates "enabled"; a value of 0 indicates "disabled".			

5.2.3 Optimized Firmware for Wireless Applications

When using the simplified configuration file format (indicated by the existence of the `ConfigVersion` parameter), if the `NumProcesses` parameter in the [WIRELESS] section of the configuration file is greater than 0, a version of the firmware optimized for small cryptography packets is automatically selected. In this case, each cryptography process consumes six rings as in the "standard" firmware case. The range for the `NumProcesses` parameter in the [WIRELESS] section is constrained in the same way as that describe in [Maximum Number of Process Calculations](#) on page 79), except that only cryptography instances (no data compression instances) are supported by the optimized firmware.

The optimized firmware operates with the following constraints and characteristics:

- SGL and Flat buffers are supported.
- The maximum supported Source/Destination payload size is 2K (where payload is either a flat buffer with a size up to 2K or the total number of bytes in flat buffers specified in SGL descriptors.
- Only rings 0-31 and rings 128-159 are use, that is, the first *two banks* in the lower and upper clusters (sets of banks), *where a bank has 16 rings*.
- There is no support for the runtime (resent) `Init AE` and `Init Ring` info messages (these messages must be sent once in the start-up phase per AE).
- Cipher Only and Auth Only (Mode0/Mode1/Mode2) processing is supported.
- TRNG (INIT/GET ENTROPY)/Compression/Asymmetric (PKE) services are not supported.
- Admin service is not supported.
- Chained (Cipher-Auth/Auth-Cipher/GCM/CCM) operation is not supported.
- Partial Cipher Only or Partial Auth Only requests are not supported.
- Nested Auth operation is not supported.
- Key generation services, such as TLS/SSL/MGF are not supported.
- Wireless image does not support virtualized environments.
- Request ordering is always enabled.

5.3 Logical Instances Section

This section allows the configuration of logical instances in each address domain (kernel space and individual user space processes). See [Hardware Assisted Rings](#) on page 28 and [Logical Instances](#) on page 23 for more information.

The address domains are in the following format:



- For the kernel address domain: [KERNEL]
- For user process address domains: [xxxxxx], where xxxxxx may be any ASCII value that uniquely identifies the user mode process.

To allow a driver to correctly configure the logical instances associated with a user process, the process must call the function `icp_sal_userStartMultiProcess`, passing the xxxxx string during process initialization. When the user space process is finished, it must call the function `icp_sal_userStop` to free resources. See [User Space Access Configuration Functions](#) on page 129 for more information.

The `NumProcesses` parameter (in the User Process section) indicates the max number of user space processes within that section name with access to instances on this device. See [icp_sal_userStartMultiProcess Usage](#) for more information.

The items that can be configured for a logical instance are:

- The name of the logical instance
- The accelerator associated with this logical instance
- The core to which the instance is affinitized (optional)

5.3.1 [KERNEL] Section

In the [KERNEL] section of the configuration file, information about the number and type of kernel instances can be defined.

The following table describes the parameters that determine the number of kernel instances for each service.

Note: The maximum number of cryptographic instances supported is 32.

Parameter	Description	Default	Range
NumberCyInstances	Specifies the number of cryptographic instances. <i>Note:</i> Depends on the number of allocations to other services.	2	0 to 32
NumberDcInstances	Specifies the number of data compression instances. <i>Note:</i> Depends on the number of allocations to other services.	1	0 to 64
<i>Note:</i> "Default" denotes the value in the configuration file when shipped.			

5.3.1.1 Cryptographic Logical Instance Parameters

The following table shows the parameters that can be set for cryptographic logical instances.

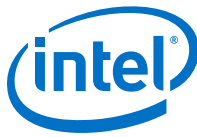


Table 8. Cryptographic Logical Instance Parameters

Parameter	Description	Default	Range
CyXName	Specifies the name of cryptographic instance number X.	IPSec0	String (max. 64 characters)
CyXAcceleratorNumber	Specifies the accelerator number that the cryptographic instance number X is assigned to.	0	0, 1, 2 or 3
CyXIsPolled	Specifies if cryptographic instance number X works in poll mode or IRQ mode.	0 for kernel space instances 1 for user space instances	For instance in the kernel space: 0 for IRQ 1 for poll mode For instance in the user space: 0 for IRQ 1 for poll mode
CyXNumConcurrentSymRequests (optional)	Specifies the number of in-progress cryptographic concurrent symmetric requests (and responses) for cryptographic instance number X. <i>Note:</i> Overrides the default <code>CyNumConcurrentSymRequests</code> value in the General section for this specific instance. <i>Note:</i> In the configuration file, this parameter must be specified before the <code>CyXCoreAffinity</code> parameter. If it is not, the default value specified in <code>CyNumConcurrentSymRequests</code> in the General section is used.	N/A	64, 128, 256, 512, 1024, 2048 or 4096
CyXNumConcurrentAsymRequests (optional)	Specifies the number of concurrent asymmetric requests for cryptographic instance number X. <i>Note:</i> Overrides the default <code>CyNumConcurrentAsymRequests</code> value in the General section for this specific instance. <i>Note:</i> In the configuration file, this parameter must be specified before the <code>CyXCoreAffinity</code> parameter. If it is not, the default value specified in <code>CyNumConcurrentAsymRequests</code> in the General section is used.	N/A	64, 128, 256, 512, 1024, 2048 or 4096
CyXCoreAffinity	Specifies the core to which the instance should be affinityized.	Varies depending on the value of X.	0 to max. number of cores in the system

Note: "Default" denotes the value in the configuration file when shipped.

5.3.1.2 Data Compression Logical Instance Parameters

The following table shows the parameters in the configuration file that can be set for data compression logical instances.

Note: The maximum number of data compression instances supported is 126.



Parameter	Description	Default	Range
DcXName	Specifies the name of data compression instance number X.	IPComp0	String (max. 64 characters)
DcXAcceleratorNumber	Specifies the accelerator number that the data compression instance number X is assigned to.	0	0 or 1
DcXIsPolled	Specifies if data compression instance number X works in poll mode or IRQ mode.	0 for kernel space instances 1 for user space instances	For instance in the kernel space: 0 for IRQ 1 for poll mode For instance in the user space: 0 for IRQ 1 for poll mode
DcXNumConcurrentRequests (optional)	Specifies the number of data compression concurrent requests. Overrides the default <code>DcNumConcurrentRequests</code> value in the General section for this specific instance. <i>Note:</i> In the configuration file, this parameter must be specified before the <code>DcXCoreAffinity</code> parameter. If it is not, the default value specified in <code>DcNumConcurrentRequests</code> in the General section is used.	N/A	64, 128, 256, 512, 1024, 2048 or 4096
DcXCoreAffinity	Specifies the core to which this data compression instance is affinitized.	Varies depending on the value of X.	0 to max. number of cores in the system
<i>Note:</i> "Default" denotes the value in the configuration file when shipped.			

5.3.2 [DYN] Section

In the [DYN] section of the configuration file, information about the number and type of instances that can be allocated dynamically are specified.

The parameters that can be included in the [DYN] section are the same as those that can be included in the [KERNEL] section. See [\[KERNEL\] Section](#) on page 75 for details.

Once the logical instances are reserved in the configuration file, they can be allocated using the dynamic instance allocation APIs. See [Dynamic Instance Allocation Functions](#) on page 107 for more information.

5.3.2.1 Dynamic Instance Configuration Example

The following configuration file snippets demonstrate the reservation of instances for dynamic allocation. In a system that uses the two configuration files below, `icp_sal_userCyInstancesAlloc` can allocate up to 26 cryptographic (cy) instances and `icp_sal_userDcInstancesAlloc` can allocate up to 14 data compression (dc) instances. See [Dynamic Instance Allocation Functions](#) on page 107 for more information.



Taken from: /etc/dh89xxcc_qa_dev0.conf

```
...

[DYN]
NumberCyInstances = 10
NumberDcInstances = 4

# Crypto - User instance DYN #0
Cy0Name = "DYN0"
Cy0IsPolled = 1
Cy0AcceleratorNumber = 0
# List of core affinities
Cy0CoreAffinity = 0

# Crypto - User instance DYN #1
CylName = "DYN1"
CylIsPolled = 1
CylAcceleratorNumber = 1
# List of core affinities
CylCoreAffinity = 1

# Crypto - User instance DYN #2
Cy2Name = "DYN2"
Cy2IsPolled = 1
Cy2AcceleratorNumber = 2
# List of core affinities
Cy2CoreAffinity = 2

...

# Data Compression - User space DYN instance #0
Dc0Name = "DCDYN0"
Dc0AcceleratorNumber = 0
Dc0IsPolled = 1
Dc0CoreAffinity = 0

# Data Compression - User space DYN instance #1
DclName = "DCDYN1"
DclAcceleratorNumber = 1
DclIsPolled = 1
DclCoreAffinity = 1

...
```

Taken from: /etc/dh89xxcc_qa_dev1.conf

```
...

[DYN]
NumberCyInstances = 16
NumberDcInstances = 10

...
```

5.3.3 User Process [xxxxx] Sections

In each [xxxxx] section of the configuration file, user space access to the device can be configured.

The following table shows the parameters in the configuration file that can be set for user process [xxxxx] sections.

**Table 9. User Process [xxxxx] Sections Parameters**

Parameter	Description	Default	Range
NumProcesses	<p>The number of user space processes with section name [xxxxx] that have access to this device.</p> <p>The maximum number of processes that can call <code>icp_sal_userStartMultiProcess</code> and be active at any one time. See icp_sal_userStartMultiProcess Usage on page 131 for more information.</p> <p>Caution: Resources are preallocated. If this parameter value is set too high, the driver fails to load.</p>	1	For constraints, see Maximum Number of Process Calculations on page 79.
LimitDevAccess	<p>Indicates if the user space processes in this section are limited to only access instances on this device.</p> <p>See Configuring Multiple PCH Devices in a System on page 80 for more information on configuring multiple user space processes on a multi device system.</p>	0	0 (disabled, processes in this section can access multiple devices) or 1 (enabled, processes in this section can only access this device)
NumberCyInstances	<p>Specifies the number of cryptographic instances.</p> <p><i>Note:</i> Depends on the number of allocations to other services.</p>	1	0 to 32
NumberDcInstances	<p>Specifies the number of data compression instances.</p> <p><i>Note:</i> Depends on the number of allocations to other services.</p>	1	0 to 126
<p><i>Note:</i> "Default" denotes the value in the configuration file when shipped.</p> <p><i>Note:</i> The order of <code>NumProcesses</code> and <code>LimitDevAccess</code> parameters is important. The <code>NumProcess</code> parameter must appear before the <code>LimitDevAccess</code> parameter in the section.</p>			

Parameters for each user process instance can also be defined. The parameters that can be included for each specific user process instance are similar to those in the [Logical Instances Section](#) on page 74.

5.3.3.1 Maximum Number of Process Calculations

The `NumProcesses` parameter is the number of user space processes per service within the [xxxx] section domain with access to this device.

The value to which this parameter can be set is determined by a number of factors, most significantly, the number of cryptography instances and/or data compression instances in the process section. The total number of processes, per service, created by the driver is given by the expression (e.g., for cryptography):

$$(\text{NumProcesses}) \times (\text{NumberCyInstances})$$

For communications between the CPU and an accelerator, each cryptography instance consumes six hardware assisted rings and each data compression instance consumes two rings. In addition, up to four rings (for each device) are reserved for administration purposes. A further constraint is that it is only possible to have two cryptography instances per bank, restricting the maximum number of cryptography instances to 32.



The total number of rings available is 256; therefore, the `NumProcesses` parameter can only be set to a value that meets the constraints described above.

The following are examples that make use of most of the rings on a device:

- `NumProcesses` can be set to 16, if `NumberCyInstances` = 2 (consuming 192 rings) and `NumberDcInstances` = 1 (consuming 32 rings), with 4 rings for administration, giving a total of 228 (meeting the <256 constraint).
- `NumProcesses` can be set to 31, if `NumberCyInstances` = 1 (consuming 186 rings) and `NumberDcInstances` = 1 (consuming 62 rings), with 4 rings for administration, giving a total of 248 (meets the <256 constraint).
- `NumProcesses` can be set to 32, if `NumberCyInstances` = 1 and `NumberDcInstances` = 0. This is because you can only have two cryptography instances per ring bank (the rings for each cryptography instance must be in the same ring bank) and there are a total of 16 banks.

Also, the following may be useful: When the `NumberXXInstances` < Number of available accelerators for that service, and `NumProcesses` ≥ Number of available accelerators, then spread the instances across the accelerators. For example, four Cy accelerators:

- 1 instance, then set `Cy0AcceleratorNumber` = 0,1,2,3
- 2 instances, then set `Cy0AcceleratorNumber` = 0,1 and `Cy1AcceleratorNumber` = 2,3 (or `Cy0AcceleratorNumber` = 0, 2 and `Cy1AcceleratorNumber` = 1,3)

5.4 Configuring Multiple PCH Devices in a System

A platform may include more than one PCH device. Each device must have its own configuration file. The format and structure of the configuration file is exactly the same for all devices. Consequently, the configuration file for device 0, `dh89xxcc_qa_dev0.conf`, can be cloned for use with other PCH devices.

Simply make a copy of the file and rename it by changing the "dev0" part of the file name, for example, for device 1 change the file name to `dh89xxcc_qa_dev1.conf`, for device 2, change the file name to `dh89xxcc_qa_dev2.conf` and so on. Then, you can configure each device by editing the corresponding configuration file accordingly. There can be up to 32 PCH devices on a platform.

Each PCH device must have its own configuration file. If a configuration file does not exist for a device, that device will not start at all and an error is displayed indicating that a configuration file was not found.

To determine the number of PCH devices in a system, use the `lspci` utility:

```
lspci -d 8086:0434
```

The output from a system with two PCH devices is similar to the following:

```
08:00.0 Co-processor: Intel Corporation Device 0434
09:00.0 Co-processor: Intel Corporation Device 0434
```




Then, after the driver is loaded, the user can use the `qat_service` script to determine the name of each device and its status. For example:

```
./qat_service status
```

Again, in a system with two PCH devices, the output looks like this:

```
icp_dev0 - type=dh89xxcc, inst_id=0, bsf=01:00:0, #accel=2, #engines=8, state=up
icp_dev1 - type=dh89xxcc, inst_id=1, bsf=05:00:0, #accel=2, #engines=8, state=up
```

The user can also use the `qat_service` to start, stop, restart and shutdown each device separately or all devices together. See [Managing Acceleration Devices Using qat_service](#) on page 41 for more information.

Some important configuration file information when using multiple PCH devices:

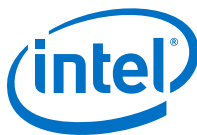
- When specifying kernel and user space instances in the configuration file, the `Cy<Number>Name` and `Dc<Number>Name` parameters must be unique in the context of the section name only. For example, it is valid to have a parameter called `Cy0Name` in both a kernel instance section and a user instance section in the same configuration file without issue. Also, the parameter names do not need to be unique at a system-wide level. For example, it is valid to have a parameter called `Cy0Name` in both the configuration file for `dev0` and the configuration file for `dev1` without issue.
- For devices with configuration files that have the same section name, for example, "SSL" and the same data in that section, is it necessary to use the `cpaCyInstanceGetInfo2()` function to distinguish between devices. The `cpaCyInstanceGetInfo2()` allows the user of the API to query which physical device a cryptography instance handle belongs to. In addition, for any application domain defined in the configuration files ([KERNEL], [SSL] and so on), a call to `cpaCyGetNumInstances()` returns the number of cryptography instances defined for that domain across all configuration files. A subsequent call to `cpaCyGetInstances()` obtains these instance handles.
- When using multiple configuration files, the `LimitDevAccess` parameter for a process must be enabled or disabled in all configuration files. The driver may not find the correct entries in the configuration file if the `LimitDevAccess` option is enabled in one configuration file and disabled in another.

5.5 Configuring Multiple Processes on a Multiple-Device System

As an example, consider a system with two PCH devices (in total, eight crypto acceleration engines, four on each device) where it is necessary to configure two user space sections. One section we identify as `SSL` and the other we identify as `IPSec`.

- For the `SSL` section, we want to configure eight processes, where each process has access to one acceleration instance.
- For the `IPSec` section, we want to configure one process with access to all eight acceleration engines.

In this scenario, the user space section of the configuration files would look like the following.



For dh89xxcc_qa_dev0.conf:

```
[SSL] #User space section name
NumProcesses=4 #There are 4 user space process with section name SSL with access
to this device
LimitDevAccess=1 # These 4 SSL user space processes only use this device
NumCyInstances=1 # Each process has access to 1 Cy instance on this device
NumDcInstances=0 # Each process has access to 0 Dc instances on this device

# Crypto - User instance #0
Cy0Name = "SSL0"
Cy0IsPolled = 1
Cy0AcceleratorNumber = 0,1,2,3
Cy0CoreAffinity = 0 # Core affinity not used for polled instance

[IPsec] #User space section name
NumProcesses=1 #There is 1 user space process with section name IPsec with access
to this device
LimitDevAccess=0 # This IPsec user space process may have access to other devices
NumCyInstances=4 # The IPsec process has access to 4 Cy instances on this device
NumDcInstances=0 # The IPsec process has access to 0 Dc instances on this device

# Crypto - User instance #0
Cy0Name = "IPSec0"
Cy0IsPolled = 1
Cy0AcceleratorNumber = 0
Cy0CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #1
Cy1Name = "IPSec1"
Cy1IsPolled = 1
Cy1AcceleratorNumber = 1
Cy1CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #2
Cy2Name = "IPSec2"
Cy2IsPolled = 1
Cy2AcceleratorNumber = 2
Cy2CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #3
Cy3Name = "IPSec3"
Cy3IsPolled = 1
Cy3AcceleratorNumber = 3
Cy3CoreAffinity = 0 # Core affinity not used for polled instance
```

For dh89xxcc_dev1.conf:

```
[SSL] #User space section name
NumProcesses=4 #There are 4 user space process with section name SSL with access
to this device
LimitDevAccess=1 # These 4 SSL user space processes only use this device
NumCyInstances=1 # Each process has access to 1 Cy instance on this device
NumDcInstances=0 # Each process has access to 0 Dc instances on this device

# Crypto - User instance #0
Cy0Name = "SSL0"
Cy0IsPolled = 1
Cy0AcceleratorNumber = 0,1,2,3
Cy0CoreAffinity = 0 # Core affinity not used for polled instance

[IPsec] #User space section name
NumProcesses=1 #There is 1 user space process with section name IPsec with access
to this device
LimitDevAccess=0 # This IPsec user space process may have access to other devices
NumCyInstances=4 # The IPsec process has access to 4 Cy instances on this device
NumDcInstances=0 # The IPsec process has access to 0 Dc instances on this device

# Crypto - User instance #0
Cy0Name = "IPSec0"
Cy0IsPolled = 1
```

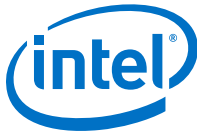


```
Cy0AcceleratorNumber = 0
Cy0CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #1
Cy1Name = "IPSec1"
Cy1IsPolled = 1
Cy1AcceleratorNumber = 1
Cy1CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #2
Cy2Name = "IPSec2"
Cy2IsPolled = 1
Cy2AcceleratorNumber = 2
Cy2CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #3
Cy3Name = "IPSec3"
Cy3IsPolled = 1
Cy3AcceleratorNumber = 3
Cy3CoreAffinity = 0 # Core affinity not used for polled instance
```

Eight processes (with section name SSL) can call the `icp_sal_userStartMultiProcess("SSL", CPA_TRUE)` function to get access to one crypto instance each. One process (with section name IPSec) can call the `icp_sal_userStartMutliProcess("IPSec", CPA_FALSE)` function to get access to eight crypto instances.

Internally in the driver, this works as follows:

1. When the driver is configured (that is, the service `qat_service` is called), the driver reads the configuration file for the device and populates an internal configuration table.
2. Reading the configuration file for dev0:
 - a. For the section named [SSL], the driver determines that four processes are required and that these processes are limited to access to this device only. In this case, the driver creates four internal sections that it labels `SSL_DEV0_INT_0`, `SSL_DEV0_INT_1`, `SSL_DEV0_INT_2` and `SSL_DEV0_INT_3`. Each section is given access to one crypto instance as described.
 - b. For section name [IPSec], the driver determines that one process is required and that this process is not limited to access to this device only (that is, it may access instances on other devices). In this case, the driver creates one internal section that it labels `IPSec_INT_0` and gives this access to four crypto instances on this device.
3. Reading the configuration file for dev1:
 - a. For the section named [SSL], the driver determines that four processes are required and that these processes are limited to access this device only. In this case, the driver creates four internal sections that it labels `SSL_DEV1_INT_0`, `SSL_DEV1_INT_1`, `SSL_DEV1_INT_2` and `SSL_DEV1_INT_3`. Each section is given access to one crypto instance as described.
 - b. For the section named [IPSec], the driver determines that one process is required and that this process may have access to instances on other devices. In this case, the driver creates one internal section that it labels `IPSec_INT_0` and gives this access to four crypto instances on this device. Notice that this section name now appears in both devices' internal configuration and therefore the process that gets assigned this section name will have access to instances on both devices.



4. In total, there are nine separate sections (SSL_DEV0_INT_0, SSL_DEV0_INT_1, SSL_DEV0_INT_2, SSL_DEV0_INT_3, SSL_DEV1_INT_0, SSL_DEV1_INT_1, SSL_DEV1_INT_2, SSL_DEV1_INT_3 and IPsec_INT_0) with access to crypto instances.

When a process calls the `icp_sal_userStartMultiProcess("SSL", CPA_TRUE)` function, the driver locates the next available section of the form `SSL_DEV<m>_INT<...>` (of which there are eight in total in this example) and assigns this section to the process. This gives the process access to corresponding crypto instances.

When a process calls the `icp_sal_userStartMultiProcess("IPsec", CPA_FALSE)` function, the driver locates the next available section of the form `IPsec_INT<...>` (of which there is only one in total for this example) and assigns this section to the process. This gives the process access to the corresponding crypto instances.

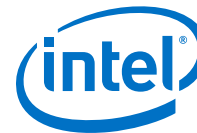
Note:

If a process calls the `icp_sal_userStartMultiProcess("IPsec", CPA_TRUE)` function, the driver locates the next available section of the form `IPsec_DEV<m>_INT<...>` and gives the process access to corresponding crypto instances (zero in this example, since `LimitDevAccess=0` in the IPsec section of the config file). When the process queries the number of crypto instances in this case (using `cpaCyGetNumInstances()`), the number returned will be zero because this process was assigned a section that was not configured with any instances using the config file.

5.6 Sample Configuration File (V2)

This following sample configuration file is provided in the software package.

```
#####
#
# @par
# This file is provided under a dual BSD/GPLv2 license. When using or
# redistributing this file, you may do so under either license.
#
# GPL LICENSE SUMMARY
#
# Copyright (c) 2007-2013 Intel Corporation. All rights reserved.
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of version 2 of the GNU General Public License as
# published by the Free Software Foundation.
#
# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin St - Fifth Floor, Boston, MA 02110-1301 USA.
# The full GNU General Public License is included in this distribution
# in the file called LICENSE.GPL.
#
# Contact Information:
# Intel Corporation
#
# BSD LICENSE
#
```



```
# Copyright(c) 2007-2013 Intel Corporation. All rights reserved.
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
# * Redistributions of source code must retain the above copyright
# notice, this list of conditions and the following disclaimer.
# * Redistributions in binary form must reproduce the above copyright
# notice, this list of conditions and the following disclaimer in
# the documentation and/or other materials provided with the
# distribution.
# * Neither the name of Intel Corporation nor the names of its
# contributors may be used to endorse or promote products derived
# from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
# A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
# OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
# SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
# LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
# version: QAT1.5.L.1.10.0-65
#####
# General Section
#####

[GENERAL]
ServicesEnabled = cy0;cy1;dc

# Use version 2 of the config file
ConfigVersion = 2
# Look Aside Cryptographic Configuration
cyHmacAuthMode = 1

# Look Aside Compression Configuration
dcTotalSRAMAvailable = 0

# Firmware Location Configuration
Firmware_MofPath = mof_firmware.bin
Firmware_MmpPath = mmp_firmware.bin

#Default values for number of concurrent requests*/
CyNumConcurrentSymRequests = 512
#The value CyNumConcurrentAsymRequests will do impact to memory
#consumption greatly. Below is some memory consumption data for
#the configuration per instance.
# 128: 10M
# 512: 40M
# 1024: 78M
# 4096: 280M
# By default, 4 kernel instances and 4 user space instances, so if
# the value is set to be 4096, for pke, the memory consumption is:
# (4+4)*280=2240M
CyNumConcurrentAsymRequests = 128
DcNumConcurrentRequests = 512

#Statistics, valid values: 1,0
statsGeneral = 1
statsDc = 1
statsDh = 1
statsDrbg = 1
```



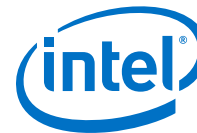
```
statsDsa = 1
statsEcc = 1
statsKeyGen = 1
statsLn = 1
statsPrime = 1
statsRsa = 1
statsSym = 1

# Enables or disables Single Root Complex IO Virtualization.
# If this is enabled (1) then SRIOV and VT-d need to be enabled in
# BIOS and there can be no Cy or Dc instances created in PF (Dom0).
# If this is disabled (0) then SRIOV and VT-d need to be disabled
# in BIOS and Cy and/or Dc instances can be used in PF (Dom0)
SRIOV_Enabled = 0

# When using virtualisation PF_bundle_offset indicates the first bundle that
# will be used to allocate instances for the Host. This and bundles
# above it will be used until all instances in below sections are allocated.
# Guests cannot share bundles with the Host so only bundles below
# and above this will be available to be assigned to VMs.
# For instance if PF_bundle_offset = 5 and there are 3 instances
# below each with different core affinities then instances in the Host
# will be allocated on bundles 5, 6 and 7 and bundles 0-4 and 8-31
# will be available for VMs.
# So if instances are needed on the Host, uncomment this and set it
# so it doesn't clash with bundles assigned to VMs.
# NOTE: bundle 0 and 8 and used for admin messages and can not be used
# for services via neither PF nor VF.
#PF_bundle_offset = 1

#Debug feature, if set to 1 it enables additional entries in /proc filesystem
ProcDebug = 1

#####
#
# Logical Instances Section
# A logical instance allows each address domain
# (kernel space and individual user space processes)
# to configure rings (i.e. hardware assisted queues)
# to be used by that address domain and to define the
# behavior of that ring.
#
# The address domains are in the following format
# - For kernel address domains
#   [KERNEL]
# - For user process address domains
#   [xxxxx]
# Where xxxxx may be any ascii value which uniquely identifies
# the user mode process.
# To allow the driver correctly configure the
# logical instances associated with this user process,
# the process must call the icp_sal_userStartMultiProcess(...)
# passing the xxxxx string during process initialisation.
# When the user space process is finished it must call
# icp_sal_userStop(...) to free resources.
# NumProcesses will indicate the maximum number of processes
# that can call icp_sal_userStartMultiProcess on this instance.
# Warning: the resources are preallocated: if NumProcesses
# is too high, the driver will fail to load
#
# Items configurable by a logical instance are:
# - Name of the logical instance
# - The accelerator associated with this logical
#   instance
# - The core the instance is affinitized to (optional)
#
# Note: Logical instances may not share the same ring, but
#       may share a ring bank.
#
# The format of the logical instances are:
# - For crypto:
```



```
#           Cy<n>Name = "xxxx"
#           Cy<n>AcceleratorNumber = 0-3
#           Cy<n>CoreAffinity = 0-7
#
# - For Data Compression
#           Dc<n>Name = "xxxx"
#           Dc<n>AcceleratorNumber = 0-1
#           Dc<n>CoreAffinity = 0-7
#
# Where:
#     - n is the number of this logical instance starting at 0.
#     - xxxx may be any ascii value which identifies the logical instance.
#
# Note: for user space processes, a list of values can be specified for
# the accelerator number and the core affinity: for example
#           Cy0AcceleratorNumber = 0,2
#           Cy0CoreAffinity = 0,2,4
# These comma-separated lists will allow the multiple processes to use
# different accelerators and cores, and will wrap around the numbers
# in the list. In the above example, process 0 will use accelerator 0,
# and process 1 will use accelerator 2
#
#####

#####
# Kernel Instances Section
#####
[KERNEL]
NumberCyInstances = 4
NumberDcInstances = 2

# Crypto - Kernel instance #0
Cy0Name = "IPSec0"
Cy0AcceleratorNumber = 0
Cy0IsPolled = 0
Cy0CoreAffinity = 0

# Crypto - Kernel instance #1
Cy1Name = "IPSec1"
Cy1AcceleratorNumber = 1
Cy1IsPolled = 0
Cy1CoreAffinity = 1

# Crypto - Kernel instance #2
Cy2Name = "IPSec2"
Cy2AcceleratorNumber = 2
Cy2IsPolled = 0
Cy2CoreAffinity = 2

# Crypto - Kernel instance #3
Cy3Name = "IPSec3"
Cy3AcceleratorNumber = 3
Cy3IsPolled = 0
Cy3CoreAffinity = 3

# Data Compression - Kernel instance #0
Dc0Name = "IPComp0"
Dc0AcceleratorNumber = 0
Dc0IsPolled = 0
Dc0CoreAffinity = 0

# Data Compression - Kernel instance #1
Dc1Name = "IPComp1"
Dc1AcceleratorNumber = 1
Dc1IsPolled = 0
#Concurrent request value can optionally be overwritten
#Dc1NumConcurrentRequests = 256
Dc1CoreAffinity = 1

#####
# User Process Instance Section
```



```
#####  
[SSL]  
NumberCyInstances = 4  
NumberDcInstances = 2  
NumProcesses = 1  
LimitDevAccess = 0  
  
# Crypto - User instance #0  
Cy0Name = "SSL0"  
Cy0IsPolled = 1  
Cy0AcceleratorNumber = 0  
# List of core affinities  
Cy0CoreAffinity = 0  
  
# Crypto - User instance #1  
Cy1Name = "SSL1"  
Cy1IsPolled = 1  
Cy1AcceleratorNumber = 1  
# List of core affinities  
Cy1CoreAffinity = 1  
  
# Crypto - User instance #2  
Cy2Name = "SSL2"  
Cy2IsPolled = 1  
Cy2AcceleratorNumber = 2  
# List of core affinities  
Cy2CoreAffinity = 2  
  
# Crypto - User instance #3  
Cy3Name = "SSL3"  
Cy3IsPolled = 1  
Cy3AcceleratorNumber = 3  
# List of core affinities  
Cy3CoreAffinity = 3  
  
# Data Compression - User space instance #0  
Dc0Name = "UserDC0"  
Dc0AcceleratorNumber = 0  
Dc0IsPolled = 1  
Dc0CoreAffinity = 0  
  
# Data Compression - User space instance #1  
Dc1Name = "UserDC1"  
Dc1AcceleratorNumber = 1  
Dc1IsPolled = 1  
Dc1CoreAffinity = 1  
  
#####  
# Wireless Process Instance Section  
#####  
[WIRELESS]  
NumberCyInstances = 1  
NumberDcInstances = 0  
NumProcesses = 0  
  
# Crypto - User instance #0  
Cy0Name = "WIRELESS0"  
Cy0IsPolled = 1  
Cy0AcceleratorNumber = 0  
# List of core affinities  
Cy0CoreAffinity = 0
```




5.7 Configuration File Version 2 Differences

Note: Both the configuration file Version 2 and Version 1 are supported by the acceleration driver. The `ConfigVersion` parameter if present and set to 2 (`ConfigVersion = 2`) indicates that the new configuration format will be used. Otherwise, the older format is used as before.

The following is a summary of the differences between the configuration file Version 2 and Version 1 file format:

- Bank and ring numbers are no longer specified in the configuration file; they are dynamically allocated.
- Core affinity can be specified for each instance. The driver will allocate a bank with that affinity.
- The number of current requests (for symmetric cryptography, asymmetric cryptography and data compression) are now specified in the General section of the configuration file, and can be overwritten for each particular instance if needed. If they are not specified at all, a default value is used by the driver.
- Accelerator number and execution engine parameters engine have been merged. The interpretation now is that there are four accelerators as opposed to two accelerators with two engines for each accelerator.
- Interrupt coalescing parameters are now in the General section (previously in the Accelerator sections).
- In the User Space section, the new `NumProcesses` parameter allows that number of processes to use that section. The core affinity for each of the processes is specified in a comma separated list.

For example, if `Cy0AcceleratorNumber=0,1,2,3`, the first process uses accelerator 0, the second uses accelerator 1, and so on. If there are more processes than list elements, it loops back. For example, if there are 8 processes and the list only contains elements 0,1,2,3, the fourth process uses accelerator 0 again, the fifth process uses accelerator 1, and so on. In order to use this functionality, the processes must be started with the [icp_sal_userStartMultiProcess](#) function.

- The `LimitDevAccess` parameter has been added. This parameter indicates if the user space processes in the section containing the `LimitDevAccess` parameter are limited to only access instances on a specific device.



6.0 Secure Architecture Considerations

This chapter describes the potential threats identified as part of the secure architecture analysis of the Acceleration Complex within the Intel® Communications Chipset 8900 to 8920 Series (PCH) and the actions that can be taken to protect against these threats. This chapter concentrates on the Acceleration Complex. There are no additional security considerations related to other major components within the PCH, including the GbE component (based on the Intel® 82580 Gigabit Ethernet Controller), and the I/O component (based on the Intel® P55 Express Chipset).

First, the terminology covering the main threat categories and mechanisms, attacker privilege and deployment models are presented. Then, some common mitigation actions that can be applied to many of these threat categories and mechanisms are discussed. Finally, more specific threat/attack vectors, including attacks against specific services of the PCH device are described.

6.1 Terminology

Each of the potential threat/attack vectors discussed may be described in terms of the following:

- [Threat Categories](#) on page 90
- [Attack Mechanism](#) on page 90
- [Attacker Privilege](#) on page 91
- [Deployment Models](#) on page 91

6.1.1 Threat Categories

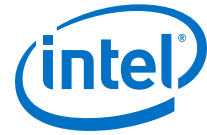
System threats can be classified into the categories in the following table.

Table 10. System Threat Categories

Category	Nature of Threat and Examples
Exposure of Data	<ul style="list-style-type: none">• Attacker reads data to which they should not have read access• Attacker reads cryptographic keys
Modification of Data	<ul style="list-style-type: none">• Attacker overwrites data to which they should not have write access• Attacker overwrites cryptographic keys
Denial of Service	<ul style="list-style-type: none">• Attacker causes application or driver software (running on an IA core) to crash• Attacker causes Intel® QuickAssist Accelerator firmware to crash• Attacker causes excessive use of resource (IA core, Intel® QuickAssist Accelerator firmware thread, silicon slice, PCIe* bandwidth, and so on), thereby reducing availability of the service to legitimate clients

6.1.2 Attack Mechanism

Some of the mechanisms by which an attacker can carry out an attack are listed in the following table.

**Table 11. Attack Mechanisms and Examples**

Mechanism	Examples
Contrived packet stream	Attacker crafts a packet stream that exploits known vulnerabilities in the software, firmware or hardware. This could include vulnerabilities such as buffer overflow bugs, lack of parameter validation, and so on.
Compromised application software	Attacker modifies the application code calling the Intel® QuickAssist Technology API to exploit known vulnerabilities in the driver/hardware.
Application Malware	In an environment where an attacker may be able to run their own application, separate from the main application software, they may invoke the Intel® QuickAssist Technology API to exploit known vulnerabilities in the driver/hardware.
Compromised IA driver software	Attacker modifies the IA driver to exploit known vulnerabilities in the driver/hardware.
Compromised Intel® QuickAssist Technology firmware	Attacker modifies the Intel® QuickAssist Technology firmware to exploit vulnerabilities in the hardware.
Compromised public key firmware <i>Note:</i> For a description of this public key firmware, and how it differs from the Intel® QuickAssist Technology firmware, see Crypto Service Threats - Modification of Public Key FW	Attacker modifies the public key firmware to exploit vulnerabilities in the hardware.
Defect	It is also possible that the attack is not malicious, but rather an unintentional defect.

6.1.3 Attacker Privilege

The following table describes the privileges that an attacker may have. The table describes the case of a non-virtualized system.

Table 12. Attacker Privilege

Privilege	Comments
Physical access	There is no attempt to protect against threats, such as signal probes, where the attacker has physical access to the system. Customers can protect their systems using physical locks, tamper-proof enclosures, Faraday cages, and so on.
Logged in as privileged user	There is no attempt to protect against threats where the attacker is logged in as a privileged user. Customers can protect their systems using strong, frequently-changed passwords, and so on.
Logged in as unprivileged user	If the attacker is logged into a platform as an unprivileged user, it is important to ensure that they cannot use the services of the PCH to access (read or write) any data to which they would not otherwise have access.
Ability to send packets	In almost all deployments, attackers have the ability to send arbitrary packets from the network (either on LAN or WAN) into the system. It is assumed that threats (for example, contrived packet streams to exploit known vulnerabilities) may arrive in this way.

6.1.4 Deployment Models

Some of the possible deployment models are given in the following table.



Table 13. Deployment Models

Deployment Model	Examples
System with no untrusted users	<ul style="list-style-type: none">• Network security appliance• Server in data center
System with potentially untrusted users	<ul style="list-style-type: none">• Server in data center

6.2 Threat/Attack Vectors

A thorough analysis has been conducted by considering each of the threat categories, attack mechanisms, attacker privilege levels, and deployment models. As a result, the following threats have been identified. Also described are the steps a user of the PCH chipset can take to mitigate against each threat.

Some general practices that mitigate many of the common threats are considered first. Thereafter, threats on specific services (such as cryptography, data compression) and mitigation against those threats are described.

6.2.1 General Mitigation

The following mitigation techniques are generic to a number of different threat and attack vectors:

- Intel follows Secure Coding guidelines, including performing code reviews and running static analysis on its driver software and firmware, to ensure its compliance with security guidelines. It is recommended that customers follow similar guidelines when developing application code. This should include the use of tools such as static analysis, fuzzing, and so on.
- Ensure each module (including the PCH chipset, processor, and DRAM) is physically secured from attackers. This can include such examples as physical locks, tamper proofing, and Faraday cages (to prevent side-channel attacks via electromagnetic radiation).
- Ensure that network services not required on the module are not operating and that the corresponding network ports are locked down.
- Use strong passwords to protect against dictionary and other attacks on administrative and other login accounts.

6.2.2 General Threats

General threats include the following:

- [DMA](#) on page 93
- [Intentional Modification of IA Driver](#) on page 93
- [Modification of Intel® QuickAssist Accelerator Firmware](#) on page 94
- [Modification of the PCH Configuration File](#) on page 94
- [Malicious Application Code](#) on page 94
- [Contrived Packet Stream](#) on page 94



6.2.2.1 DMA

Threat: The PCH can perform Direct Memory Access (DMA, the copying of data) between arbitrary memory locations, without any of the processor's normal memory protection mechanisms. Once an attacker has sufficient privilege to invoke the Intel® QuickAssist Technology API, or to write to/read from the hardware rings used by the driver to communicate with the device, they can send requests to the Intel® QuickAssist Accelerator to perform such DMA, passing arbitrary physical memory addresses as the source and/or destination addresses, thereby reading from and/or writing to regions of memory to which they would otherwise not have access.

Mitigation: Ensure that only trusted users are granted permissions to access the Intel® QuickAssist Technology API, or to write to and read from the hardware rings. Specifically, the PCH configuration file describes logical instances of acceleration services and the set of hardware rings to be used for each such instance. User processes can ask the kernel driver to map these rings into their address spaces. To access a given device (identified by the number <N> in the filenames below), the user must be granted read/write access to the following files, which may be in /dev or /dev/icp_mux:

- icp_dev<N>_csr
- icp_dev<N>_ring
- icp_dev_mem
- icp_dev_mem_page
- icp_dev_processes

The recommendation is that these files have the following permissions by default¹:

```
# ls -l /dev/icp_dev0_ring
crw-----. 1 root root 249, 0 Jan 17 16:01 /dev/icp_dev0_ring
```

To grant permission to a given user to use the API, that user should be given membership of a group, e.g. group “adm”, and the group ownership and permissions should be changed to the following:

```
# ls -l /dev/icp_dev0_ring
crw-rw----. 1 root adm 249, 0 Jan 17 16:02 /dev/icp_dev0_ring
```

Such permissions and group membership should only be provided to trusted users. Such user accounts should be protected with strong passwords.

6.2.2.2 Intentional Modification of IA Driver

Threat: An attacker can potentially modify the IA driver to behave maliciously.

Mitigation: The driver object/executable file on disk should be protected using the normal file protection mechanisms so that it is writable only by trusted users, for example, a privileged user or an administrator.

¹ Permissions shown only for one file, but these apply to all files listed.



6.2.2.3 Modification of Intel® QuickAssist Accelerator Firmware

Threat: An attacker can potentially modify the Intel® QuickAssist Accelerator firmware to behave maliciously. The attacker can then attempt to overwrite the firmware image on disk (so that it gets downloaded on future reboots) or to download the malicious firmware image after the original image has been downloaded, thereby overwriting it.

Mitigation: The firmware image on disk should be protected using normal file protection mechanisms so that it is writable only by trusted users, for example, a privileged user or an administrator.

The implementation of the API for downloading firmware to the Intel® QuickAssist Accelerator requires access to a special administrative hardware ring. See the mitigation for the [DMA](#) on page 93 threat to limit access to this ring.

6.2.2.4 Modification of the PCH Configuration File

Threat: The PCH configuration file is read at initialization time by the driver and specifies what instances of each service (cryptographic, data compression) should be created, and which rings each service instance will use. Modifying this file could lead to denial of service (by deleting required instances), or could be used to attempt to create additional instances that the attacker could subsequently attempt to access for malicious purposes.

Mitigation: The configuration file should be protected using the normal file protection mechanisms so that it is writable only by trusted users, for example, a privileged user or an administrator.

Note: By default, the configuration file is stored in the `/etc` directory and may be named something like, `dh89xxcc_qa_dev0.conf`. Its default permissions are that it is readable and writeable only by root.

6.2.2.5 Malicious Application Code

Threat: An attacker who can gain access to the Intel® QuickAssist Technology API may be able to exploit the following features of the API:

- Simply sending requests to the accelerator at a high rate reduces the availability of the service to legitimate users.
- Buffers passed to the API have a specified length of up to 32 bits. By specifying excessive lengths, an attacker may be able to cause denial of service by overwriting data beyond the end of a buffer.
- Buffer lists passed to the API consist of a scatter gather list (array of buffers). An attacker may incorrectly specify the number of buffers, causing denial of service due to the reading or writing of incorrect buffers.

Mitigation: Only trusted users should be allowed to access the Intel® QuickAssist Technology API, as described as part of the Mitigation threat for the [DMA](#) on page 93.

6.2.2.6 Contrived Packet Stream

Threat: An attacker may attempt to contrive a packet stream that monopolizes the acceleration services, thereby denying service to legitimate users. This may consist of one or more of the following:



- Sending packets that are compressed (for example, using IPComp) or encrypted (for example, using IPsec), thereby reducing the availability of these services to legitimate traffic.
- Sending excessively large packets, causing some latency for legitimate packets.
- Sending small packets at a high packet rate, causing extra bandwidth utilization on the PCI Express* bus connecting the device to the processor.

Mitigation: Depending on the deployment scenario, it is usually not possible to prevent such attempts at denial of service. The system should be designed to cope with the worst case in terms of throughput and latency at all packet sizes.

6.2.3 Threats Against the Cryptographic Service

Threats against the cryptographic service include:

- [Reading and Writing of Cryptographic Keys](#) on page 95
- [Modification of Public Key Firmware](#) on page 95
- [Failure of the Entropy Source for the Random Number Generator](#) on page 96
- [Interference Among Users of the Random Number Service](#) on page 96

6.2.3.1 Reading and Writing of Cryptographic Keys

Threat: Cryptographic keys are stored in DRAM. An attacker who can determine where these are stored could read the DRAM to get access to the keys, or could write the DRAM to use keys known by the attacker, thereby compromising the confidentiality of data protected by these keys.

Mitigation: DRAM is considered to be inside the cryptographic boundary (as defined by FIPS 140-2). The normal memory protection schemes provided by the Intel® architecture processor and memory controller, and by the operating system, prevent unauthorized access to these memory regions.

6.2.3.2 Modification of Public Key Firmware

Background: In addition to the Intel® QuickAssist Accelerator firmware which is downloaded to the Acceleration Complex within the PCH by the driver at initialization time, there is a library of small public key firmware routines, one of which is downloaded to the device along with each request to perform a public key cryptographic primitive, such as an RSA sign operation. This public key firmware is part of the driver image (on disk), and is stored in DRAM at run-time so that it can be downloaded to the device when required.

Threat: An attacker can potentially modify the public key firmware to behave maliciously. For this to be useful, they must overwrite the firmware image on disk (so that it gets read into DRAM at initialization time on future reboots) or in DRAM (so that it gets downloaded with future PKE requests).

Mitigation: The public key firmware image on disk should be protected using normal file protection mechanisms so that it is writable only by trusted users, for example, a privileged user or an administrator. The public key firmware image in DRAM is accessible only to the process/context in which it is executing, and sending the image to the Intel® QuickAssist Accelerator requires permission to use the API and write to the corresponding hardware ring. See the mitigation for the DMA threat to limit access to such rings.



6.2.3.3 Failure of the Entropy Source for the Random Number Generator

Threat: The PCH has a non-deterministic random bit generator (NRBG, aka True Random Number Generator or TRNG) implemented in silicon that can be used as an entropy source for a deterministic random bit generator (DRBG, aka Pseudo Random Number Generator or PRNG). A failure of the entropy source can lead to poor quality random numbers, which can compromise the security of the system.

Mitigation: The NRBG has a built-in self test that detects repeated sequences of bits. A failure of the entropy source is indicated to the application/user via calls to the API. It is the responsibility of the application to decide whether and when to fail the module as a result of a failed entropy source.

6.2.3.4 Interference Among Users of the Random Number Service

Threat: The original API for random number generation (in `cpa_cy_rand.h` file, as delivered as part of an earlier generation of the Intel® QuickAssist Accelerator) had a single instance of the DRBG that was shared by all users. An attacker with appropriate permissions to access the DRBG service in one process/address space could re-seed the DRBG and thereby modify the subsequent outputs of the DRBG in other processes or contexts.

Mitigation: The API has been updated for the current generation. The updated API (`cpa_cy_drbg.h`) supports a FIPS-compliant DRBG API with multiple instances. Re-seeding one such instance does not interfere with the output of another instance. The original API has been deprecated. Applications should use the new API.

6.2.4 Data Compression Service Threats

Threats against the Data Compression service include:

- [Read/Write of Save/Restore Context](#) on page 96
- [Stateful Behavior](#) on page 96
- [Incomplete or Malformed Huffman Tree](#) on page 97
- [Contrived Packet Stream](#) on page 97

6.2.4.1 Read/Write of Save/Restore Context

Threat: The save/restore context is stored in DRAM. An attacker may attempt to read this memory to determine information about the packet stream. An attacker may also overwrite this context, affecting the result of the compression/decompression.

Mitigation: DRAM is considered to be inside the cryptographic boundary (as defined by FIPS 140-2). The normal memory protection schemes provided by the Intel® architecture processor and memory controller, and by the operating system, prevent unauthorized access to these memory regions.

6.2.4.2 Stateful Behavior

Threat: The combination of stateful behavior and requests to compress/decompress small regions of memory can lead to reduced significant overhead, and could potentially be exploited as part of a denial of service attack. This is because stateful contexts requires that the service restore and save the session state for each request. The session state includes history data and can be significantly larger than the packet, especially for small packets.



Mitigation: To minimize this overhead, the application can use stateless sessions.

6.2.4.3 Incomplete or Malformed Huffman Tree

Threat: An attacker who can run malicious code on the platform (see [Malicious Application Code](#) on page 94) can deny service (reduce performance) by sending in a rogue request with an incomplete or malformed Huffman tree. A transmission error may also lead to this situation occurring.

Mitigation: See the mitigation proposed in [Malicious Application Code](#) on page 94. Furthermore, the slice detects such incomplete or malformed Huffman trees and returns an error.

6.2.4.4 Contrived Packet Stream

Threat: Similar to the general attack mechanism described in [Contrived Packet Stream](#) on page 94, there are some aspects that are specific to the data compression service:

- An attacker can craft a compressed packet stream with a very large compression ratio (for example, 1000:1). Generating an output buffer that is significantly larger than the input buffer may reduce availability of the service to legitimate clients.
- An attacker can craft a packet stream with a large number of zero-length deflate blocks. This causes the slice to consume input, but produce no output.

Mitigation: The output is limited to the size of output buffer. Buffer exhaustion detection is built into the hardware. Therefore, the application developer should allocate output buffers based on the largest compression ratio that they wish to deal with, as required by the application or protocol, and then handle errors reported by the API.



7.0 Supported APIs

The supported APIs are described in two categories:

- [Intel® QuickAssist Technology APIs](#) on page 98
- [Additional APIs](#) on page 106

7.1 Intel® QuickAssist Technology APIs

The platforms described in this manual supports the following Intel® QuickAssist Technology API libraries:

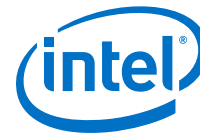
- Cryptographic - API definitions are located in: `$ICP_ROOT/quickassist/include/lac`, where `$ICP_ROOT` is the directory where the Acceleration software is unpacked. See the *Intel® QuickAssist Technology Cryptographic API Reference Manual* for details.
- Data Compression - API definitions are located in: `$ICP_ROOT/quickassist/include/dc`. See the *Intel® QuickAssist Technology Data Compression API Reference Manual* for details.

Base API definitions that are common to the API libraries are located in: `$ICP_ROOT/quickassist/include`. See also the *Intel® QuickAssist Technology API Programmer's Guide* for guidelines and examples that demonstrate how to use the APIs.

7.1.1 Intel® QuickAssist Technology API Limitations

The following limitations apply when using the Intel® QuickAssist Technology APIs on the platforms described in this manual:

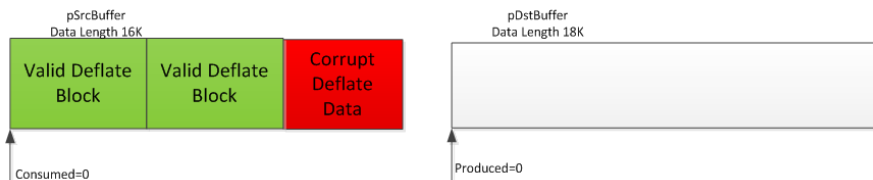
- For all services, the maximum size of a single *perform* request is 4 GB.
- For all services, data structures that contain data required by the Intel® QuickAssist Accelerator should be on a 64 Byte-aligned address to maximize performance. This alignment helps minimize latency when transferring data from DRAM to an accelerator integrated in the PCH device.
- For the key generation cryptographic API, the following limitations apply:
 - Secure Sockets Layer (SSL) key generation opdata:
 - Maximum secret length is 512 bytes
 - Maximum userLabel length is 136 bytes
 - Maximum generatedKeyLenInBytes is 248
 - Transport Layer Security (TLS) key generation opdata:
 - Secret length must be <128 bytes for TLS v1.0/1.1; <512 bytes for TLS v1.2
 - userLabel length must be <256 bytes
 - Maximum seed size is 64 bytes



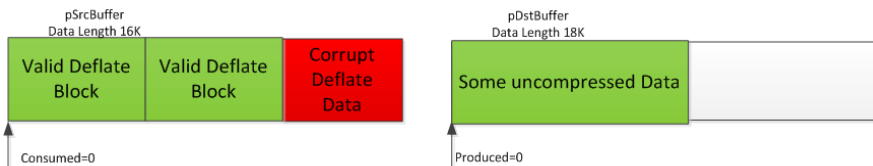
- Maximum generatedKeyLenInBytes is 248 bytes
- Mask Generation Function (MGF) opdata:
 - Maximum seed length is 255 bytes
 - Maximum maskLenInBytes is 65528
- For the cryptographic service, SNOW 3G and KASUMI operations are not supported when *CpaCySymPacketType* is set to `CPA_CY_SYM_PACKET_TYPE_PARTIAL`. The error returned in this case is `CPA_STATUS_INVALID_PARAM`.
- For the cryptographic service, when using the Deterministic Random Bit Generator (DRBG), only one in-flight request per each instantiated DRBG (that is, per each DRBG session) is allowed. If the user calls the *cpaCyDrbgGen* or *cpaCyDrbgReseed* function with the session handle of a session for which a previous request is still being processed, `CPA_STATUS_RETRY` is returned.
- For the cryptographic service, when using DRBG with a derivation function, the maximum security strength with which the DRBG can be instantiated is `CPA_CY_RBG_SEC_STRENGTH_128`. In such a case, if the user tries to instantiate DRBG with a higher security strength, the `CPA_STATUS_INVALID_PARAM` is returned.
- For the cryptographic service, when using DRBG, the requirement for the use of the derivation function (DF) is not expected to change once DRBG is instantiated.
- For the cryptographic service, when using the asymmetric crypto APIs, the buffer size passed to the API should be rounded to the next power of 2, or the next 3-times a power of 2, for optimum performance.
- For the data compression service, only one outstanding compression request per stateful session is allowed.
- For the data compression service, the size of all stateful decompression requests have to be a multiple of two with the exception of the last request.
- For the data compression service, the *CpaDcFileType* field in the *CpaDcSessionSetupData* data structure is ignored (previously this was considered for semi-dynamic compression/decompression).
- For static compression, the maximum expansion during compression is ceiling $(9 * \text{Total_Input_Byte} / 8) + 7$ bytes. If `CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_STORED_HDRS` or `CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_NO_HDRS` is selected, the maximum expansion during compression is the input buffer size plus up to ceiling $(\text{Total_Input_Byte} / 65535) * 5$ bytes, depending on whether the stored headers are selected. Note, however, due to the need for a skid pad and the way the checksum is calculated in the stored block case to prevent compression overflow, an output buffer size of ceiling $(9 * \text{Total_Input_Byte} / 8) + 55$ bytes needs to be supplied (even though the stored block output size might be less).
- The decompression service can report various error conditions most of which arise from processing dynamic Huffman code trees that are ill-formed. These soft error conditions are reported at the the Intel® QuickAssist Technology API using the *CpaDcReqStatus* enumeration. At the point of soft error, the hardware state will not be accurate to allow recovery. Therefore, in this case, the Intel® QuickAssist Technology software rolls back to the previous known good state and reports that no input has been processed and no output produced. This allows an application to correct the source of the error and resubmit the request.



For example, if the following source and destination buffers were submitted to the Intel® QuickAssist Technology:



The result would be:



- For stateful compression, the maximum output size is 4 GB. Stateful compression requests that would generate an output size greater than 4.29 GB (2^{32} bytes) will fail without an error.
- For stateful decompression, the maximum output size is 4.29 GB (2^{32} bytes).

7.1.1.1 Resubmitting After Getting an Overflow Error

The following table describes the behavior of the Intel® QuickAssist Technology compression service when an overflow occurs during a compress or decompress operation.

Table 14. Compression/Decompression Overflow Behavior

Stateful/ Stateless	Static/ Dynamic	Overflow	Input data consumed?	Valid data in output buffer?	Status Returned
Stateful (see details below)	Both	Yes	Possibly	Possibly	-11
Stateless (see details below)	Both	Yes	No	No	-11

The following describes the expected behavior of an application when an overflow occurs.

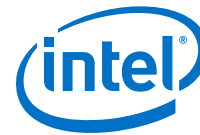
Stateful

The produced and consumed values must be used to determine where the next request starts. Internally, the session stores the cumulativeConsumedBytes and corresponding cumulative checksum based on these values and so expects the next request to continue after the valid data.

Procedure

Save the output data from the Destination buffer based on cpaDcRqResults.produced

Submit the next request with the following data:



- The first "cpaDcRqResults.consumed" bytes in the Source buffer have already been compressed, so rework the Source bufferList to start at the byte after this. Consumed = zero is a valid case; in this case, the full Source buffer must be resubmitted.
- The same Destination buffer can be re-used. It may now be big enough if part of the source data has been consumed already. Or increase if preferred.
- The results buffer can be re-used without change. In the Stateful case, the driver ignores everything in it and overwrites it on each API call.

Stateless

In the Stateless case, the entire compression request must be resubmitted with a larger output buffer. In this case, cpaDcRqResults.consumed, .produced and .checksum should be ignored. If length and checksum are required, these are not maintained in the session, and the responsibility to track these is passed up to the application.

Procedure

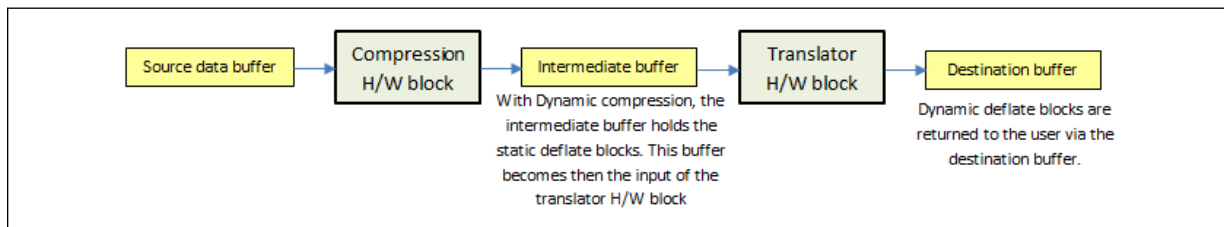
Resubmit the request with the following data:

- Use the same Source buffer.
- Allocate a bigger Destination buffer.
- Put the checksum from the previous successful request into the cpaDcRqResults struct.

7.1.1.2 Dynamic Compression for Data Compression Service

Dynamic compression involves feeding the data produced by the compression hardware block to the translator hardware block. The following figure shows the dynamic compression data path.

Figure 22. Dynamic Compression Data Path



When the compression service returns an exception (e.g., overflow error) to the user, it is recommended to examine the bytes consumed and returned in the CpaDcRqResults structure to verify if all the data in the source data buffer has been processed.

When the application selects the Huffman type to CPA_DC_HT_FULL_DYNAMIC in the session and auto select best feature is set to CPA_DC_ASB_DISABLED, the compression service may not always produce a deflate stream with dynamic Huffman trees. For example, in the case of an overflow during dynamic compression, static data will be returned in the destination buffer.



7.1.1.3 Maximal Expansion with Auto Select Best Feature for Data Compression Service

Some input data may lead to a lower than expected compression ratio. This is because the input data may not be very compressible. To achieve a maximum compression ratio, the acceleration unit provides an auto select best (ASB) feature. In this mode, the Intel® QuickAssist Technology hardware will first execute static compression followed by dynamic compression and then select the output which yields the best compression ratio. To use the ASB feature, configure the `autoSelectBestHuffmanTree` enum during the session creation.

Regardless of the ASB setting selected, dynamic compression will only be attempted if the session is configured for dynamic compression.

There are four possible settings available for the `autoSelectBestHuffmanTree` when creating a session. Based on the ASB settings described below, the produced data returned in the `CpaDcRqResults` structure will vary:

- `CPA_DC_ASB_DISABLED` - ASB mode is disabled.
- `CPA_DC_ASB_STATIC_DYNAMIC`

Both dynamic and static compression operations are performed. The size of produced data returned in the `CpaDcRqResults` structure will be the minimal value of the two operations.

```
Produced data in bytes = Min (Static, Dynamic)
```

- `CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_STORED_HDRS`

Both a dynamic and a static compression operation are performed. However, if the produced data both for the dynamic and static operations return a greater value than the uncompressed source data and source block headers, the source data will be used as a stored block. With this ASB setting, a 5-byte stored block header is prepended to the stored block.

The worst-case produced data can be estimated to:

```
Produced data in bytes = Total input bytes + ceil (Total input bytes / 65535) * 5
```

e.g., for an input source size of 111261 bytes, the worst-case produced data will be:

```
Produced data = 111261 + ceil (111261 / 65535) * 5
               = 111261 + ceil (1.698) * 5
               = 111261 + 2 * 5
Produced data = 111271 bytes
```

- `CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_NO_HDRS`

With this ASB setting, both a dynamic and a static compression operation are performed. However, if the produced data both for the dynamic and static operation return a greater value than the uncompressed source data, the uncompressed source data will be sent to the destination buffer through DMA transfer. This is the same behavior as with the ASB setting



CPA_DC_ASB_UNCOMP_STATIC_DYNAMIC_WITH_STORED_HDRS except the stored block deflate headers are not prepended to the stored block. The produced data can be estimated via the following:

```
Produced data in bytes = Min(Static, Dynamic, Uncompressed)
```

7.1.1.4 Maximal Expansion and Destination Buffer Size

For static compression operations, the worst-case possible expansion can be expressed as:

```
Max Static Produced data in bytes = ceil(9 * Total input bytes / 8) + 7
```

The memory requirement for the destination buffer is expressed by the following formula:

```
Destination buffer size in bytes = ceil(9 * Total input bytes / 8) + 55 bytes
```

The destination buffer size must take into account the worst-case possible maximal expansion + 55 bytes; e.g., for an input source size of 111261 bytes, the worst-case produced data will be:

```
Static Produced data = ceil(9 * 111261 / 8) + 7
                     = ceil (125168.625) + 7
                     = 125169 + 7
Worst case Static Produced data = 125176 bytes
Memory required for destination buffer = ceil(9 * 111261 / 8) + 55
                                     = ceil (125168.625) + 55
                                     = 125169 + 7
                                     = 125169 + 55
                                     = 125224 bytes to be allocated
```

Note: Regardless of the ASB settings, the memory must be allocated for the worst case. If an overflow occurs, either from static or dynamic compression, then the returned counters, status, and expected application behavior is as shown per the table in [Resubmitting After Getting an Overflow Error](#) on page 100.

7.1.2 Data Plane APIs Overview

The *Intel® QuickAssist Technology Cryptographic API Reference Manual* and the *Intel® QuickAssist Technology Data Compression API Reference Manual* mentioned previously contain information on the APIs that are specific to data plane applications.

These APIs are intended for use in user space applications that take advantage of the functionality provided of the Intel® Data Plane Development Kit (Intel® DPDK). The APIs are recommended for applications that are executing in a data plane environment where the cost of offload (that is, the cycles consumed by the driver sending requests to the hardware) needs to be minimized. To minimize the cost of offload, several constraints have been placed on the APIs. If these constraints are too restrictive for your application, the traditional APIs can be used instead (at a cost of additional IA cycles).



The definition of the Cryptographic Data Plane APIs are contained in:

```
$ICP_ROOT/quickassist/include/lac/cpa_cy_sym_dp.h
```

The definition of the Data Compression Data Plane APIs are contained in:

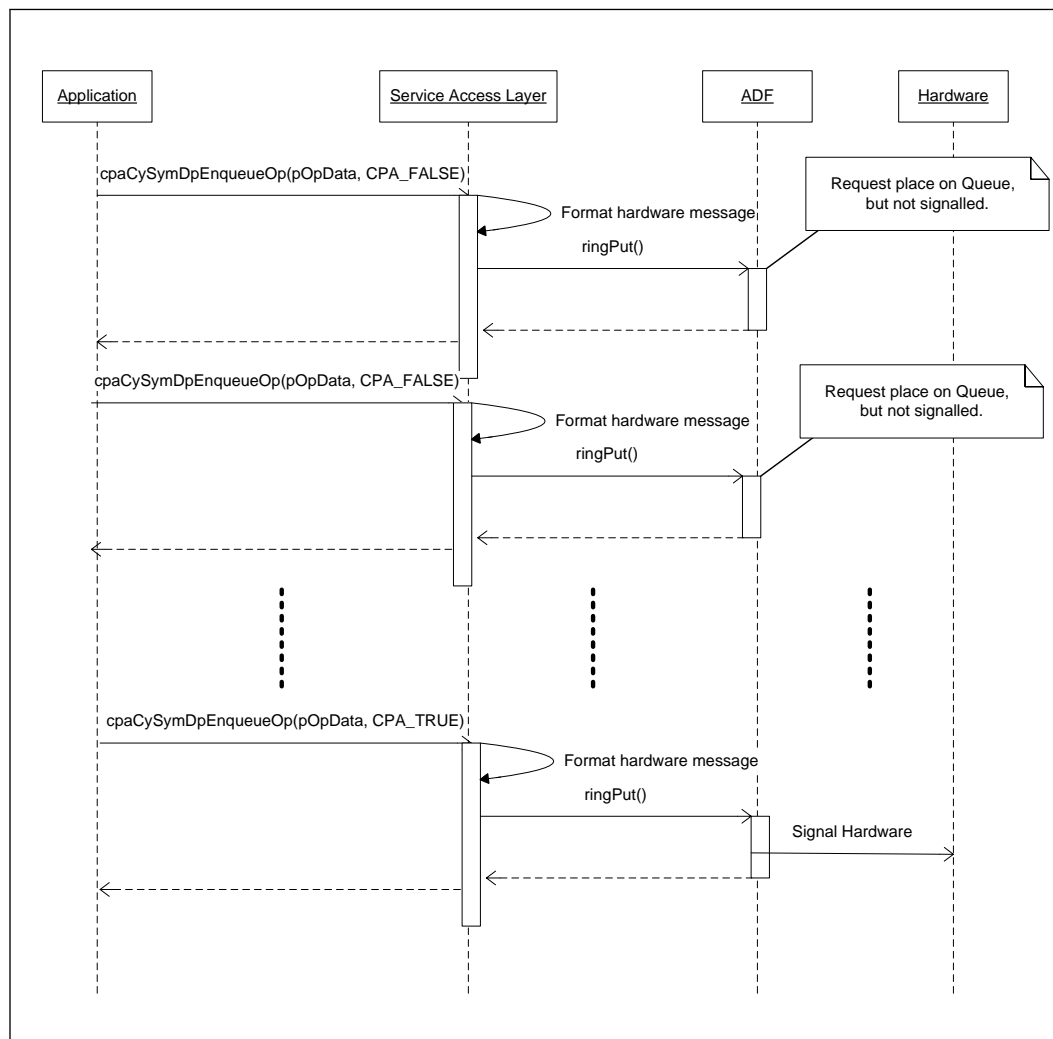
```
$ICP_ROOT/quickassist/include/dc/cpa_dc_dp.h
```

7.1.2.1 IA Cycle Count Reduction When Using Data Plane APIs

From an IA cycle count perspective, the Data Plane APIs are more performant than the traditional APIs (that is, for example, the symmetric cryptographic APIs defined in `$ICP_ROOT/quickassist/include/lac/cpa_cy_sym.h`). The majority of the cycle count reduction is realized by the reduction of supported functionality in the Data Plane APIs and the application of constraints on the calling application (see [Usage Constraints on the Data Plane APIs](#) on page 105).

In addition, to further improve performance, the Data Plane APIs attempt to amortize the cost of a Memory Mapped IO (MMIO) access when sending requests to, and receiving responses from, the hardware.

A typical usage is to call the `cpaCySymDpEnqueueOp()` or the `cpaDcDpEnqueueOp()` function multiple times with requests to process and the `performOpNow` flag set to `CPA_FALSE`. Once multiple requests have been enqueued, the `cpaCySymDpEnqueueOp()` or `cpaDcDpEnqueueOp()` function may be called with the `performOpNow` flag set to `CPA_TRUE`. This sends the requests to the Intel® QuickAssist Accelerator for processing. This sequence is shown in the following figure.


Figure 23. Amortizing the Cost of an MMIO Across Multiple Requests


The Intel® QuickAssist Technology API returns a CPA_STATUS_RETRY when the ring becomes full.

The number of requests to place on the ring is application dependent and it is recommended that performance testing be conducted with tuneable parameter values.

Two functions, `cpaCySymDpPerformOpNow()` and `cpaDCDpPerformOpNow()` are also provided that allow queued requests to be sent to the hardware without the need for queuing an additional request. This is typically used in the scenario where a request has not been received for some time and the application would like the enqueued requests to be sent to the hardware for processing.

7.1.2.2 Usage Constraints on the Data Plane APIs

The following constraints apply to the use of the Data Plane APIs. If the application can handle these constraints, the Data Plane APIs can be used:



- Thread safety is not supported. Each software thread should have access to its own unique instance (`CpaInstanceHandle`) to avoid contention on the hardware rings.
- For performance, polling is supported, as opposed to interrupts (which are comparatively more expensive). Polling functions (see [Polling Functions](#) on page 117) are provided to read responses from the hardware response queue and dispatch callback functions.
- Buffers and buffer lists are passed using physical addresses to avoid virtual-to-physical address translation costs.
- Alignment restrictions are placed on the operation data (that is, the `CpaCySymDpOpData` structure) passed to the Data Plane API. The operation data must be at least 8-byte aligned, contiguous, resident, DMA-accessible memory.
- Only asynchronous invocation is supported, that is, synchronous invocation is *not* supported.
- There is no support for cryptographic partial packets. If support for partial packets is required, the traditional Intel® QuickAssist Technology APIs should be used.
- Since thread safety is *not* supported, statistic counters on the Data Plane APIs are not atomic.
- The *default* instance (`CPA_INSTANCE_HANDLE_SINGLE`) is not supported by the Data Plane APIs. The specific handle should be obtained using the instance discovery functions (`cpaCyGetNumInstances()`, `cpaCyGetInstances()`, and `cpaDcGetNumInstances()`, `cpaDcGetInstances()`).
- The submitted requests are always placed on the high-priority ring.

7.1.2.3 Cryptographic and Data Compression API Descriptions

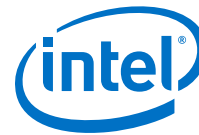
Full descriptions of the Intel® QuickAssist Technology APIs are contained in the *Intel® QuickAssist Technology Cryptographic API Reference Manual* and the *Intel® QuickAssist Technology Data Compression API Reference Manual*. In addition to the Intel® QuickAssist Technology Data Plane APIs, there are a number of Data Plane Polling APIs that are described in [Polling Functions](#) on page 117.

7.2 Additional APIs

There are a number of additional APIs that can serve for optimization and other uses outside of the Intel® QuickAssist Technology services.

These APIs are grouped into the following categories:

- [Dynamic Instance Allocation Functions](#) on page 107
- [IOMMU Remapping Functions](#) on page 115
- [Polling Functions](#) on page 117
- [Random Number Generation Functions](#)
- [User Space Access Configuration Functions](#) on page 129
- [Version Information Function](#) on page 133
- [User Space Heartbeat Functions](#) on page 132
- [Reset Device Function](#) on page 134
- [Thread-less APIs](#) on page 135



7.2.1 Dynamic Instance Allocation Functions

These functions are intended for the dynamic allocation of instances in user space. The user can use these functions to allocate/free instances defined in the [DYN] section of the configuration file.

These functions are useful if the user needs to dynamically allocate/free cryptographic (cy) or data compression (dc) instances at runtime. This is in contrast to statically specifying the number of cy or dc instances at configuration time, where the number of instances cannot be changed unless the user modifies the `.conf` file and restarts the acceleration service.

The advantage of using these functions is that the number of cy/dc instances can be changed on-demand at runtime. The disadvantage is that runtime performance is impacted if the number of cy/dc instances is changed frequently.

If the user space application knows the number of instances to be used before starting, then the user can define *Number<Service>Instances* in the [User Process] section of the `*.conf` file.

If the user space application can only know the number of instances at runtime, or wants to change the number at runtime, then the user can call the Dynamic Instance Allocation functions to allocate/free instances dynamically. The *Number<Service>Instances* in the [DYN] section of the `.conf` file(s) defines the maximum number of instances that can be allocated by user processes.

This can be useful when sharing instances among multiple applications at runtime. The maximum number of instances in a system is known in advance and it is possible to distribute them statically between applications using the configuration files. Once the driver is started, however, this cannot be changed. If, for example, there are 32 cy instances and we need to provision 16 processes, we can statically assign two cy instances per process. This can be a problem when a process needs more instances at any given time. With dynamic instance allocation, we can create a pool of instances that can be "shared" between the processes.

Continuing the example above with 32 cy instances and 16 processes, we can assign statically one cy instance to each process and create a pool of 16 [DYN] instances from the remainder. If at runtime one process needs more acceleration power, it can allocate some more instances from the pool, say, for example, eight, use them as appropriate and free them back to the pool when the work has been completed. Thereafter, other processes can use these instances as needed.

All dynamic instance allocation function definitions are located in: `$ICP_ROOT/quickassist/lookaside/access_layer/include/icp_sal_user.h`

The dynamic instance allocation functions include:

- [icp_sal_userCyGetAvailableNumDynInstances](#) on page 108
- [icp_sal_userDcGetAvailableNumDynInstances](#) on page 108
- [icp_sal_userCyInstancesAlloc](#) on page 109
- [icp_sal_userDcInstancesAlloc](#) on page 109
- [icp_sal_userCyFreeInstances](#) on page 110
- [icp_sal_userDcFreeInstances](#) on page 110
- [icp_sal_userCyGetAvailableNumDynInstancesByDevPkg](#) on page 111



- [icp_sal_userDcGetAvailableNumDynInstancesByDevPkg](#) on page 112
- [icp_sal_userCyInstancesAllocByDevPkg](#) on page 112
- [icp_sal_userDcInstancesAllocByDevPkg](#) on page 113
- [icp_sal_userCyGetAvailableNumDynInstancesByPkgAccel](#) on page 114

7.2.1.1 **icp_sal_userCyGetAvailableNumDynInstances**

Get the number of cryptographic instances that can be dynamically allocated using the `icp_sal_userCyInstancesAlloc` function.

Syntax

```
CpaStatus icp_sal_userCyGetAvailableNumDynInstances ( Cpa32U  
*pNumCyInstances );
```

Parameters

***pNumCyInstances** A pointer to the number of cryptographic instances available for dynamic allocation.

Return Value

The `icp_sal_userCyInstancesAlloc` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully retrieved the number of cryptographic instances available for dynamic allocation.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.

7.2.1.2 **icp_sal_userDcGetAvailableNumDynInstances**

Get the number of data compression instances that can be dynamically allocated using the `icp_sal_userDcInstancesAlloc` function.

Syntax

```
CpaStatus icp_sal_userDcGetAvailableNumDynInstances ( Cpa32U  
*pNumDcInstances );
```

Parameters

***pNumDcInstances** A pointer to the number of data compression instances available for dynamic allocation.

Return Value

The `icp_sal_userDcGetAvailableNumDynInstances` function returns one of the following codes:



Code	Meaning
<i>CPA_STATUS_SUCCESS</i>	Successfully retrieved the number of cryptographic instances available for dynamic allocation.
<i>CPA_STATUS_FAIL</i>	Indicates a failure.

7.2.1.3 icp_sal_userCyInstancesAlloc

Allocate the specified number of cryptographic (cy) instances from the amount specified in the [DYN] section of the configuration file. The *numCyInstances* parameter specifies the number of cy instances to allocate and must be less than or equal to the value of the *NumberCyInstances* parameter in the [DYN] section of the configuration file.

Syntax

```
CpaStatus icp_sal_userCyInstancesAlloc ( Cpa32U numCyInstances,
CpaInstanceHandle *pCyInstances);
```

Parameters

numCyInstances The number of cy instances to allocate.

**pCyInstances* A pointer to the cy instances.

Return Value

The *icp_sal_userCyInstancesAlloc* function returns one of the following codes:

Code	Meaning
<i>CPA_STATUS_SUCCESS</i>	Successfully allocated the sepecified number of cy instances.
<i>CPA_STATUS_FAIL</i>	Indicates a failure.

7.2.1.4 icp_sal_userDcInstancesAlloc

Allocate the specified number of data compression (dc) instances from the amount specified in the [DYN] section of the configuration file. The *numDcInstances* parameter specifies the number of dc instances to allocate and must be less than or equal to the value of the *NumberDcInstances* parameter in the [DYN] section of the configuration file.

Syntax

```
CpaStatus icp_sal_userDcInstancesAlloc ( Cpa32U numDcInstances,
CpaInstanceHandle *pDcInstances);
```

Parameters

numDcInstances The number of dc instances to allocate.



***pDcInstances** A pointer to the dc instances.

Return Value

The `icp_sal_userDcInstancesAlloc` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully allocated the specified number of dc instances.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.

7.2.1.5 `icp_sal_userCyFreeInstances`

Free the specified number of cryptographic (cy) instances from the amount specified in the [DYN] section of the configuration file. The *numCyInstances* parameter specifies the number of cy instances to free.

Syntax

```
CpaStatus icp_sal_userCyFreeInstances ( Cpa32U numCyInstances,  
CpaInstanceHandle *pCyInstances );
```

Parameters

numCyInstances The number of cy instances to free.

***pCyInstances** A pointer to the cy instances to free.

Return Value

The `icp_sal_userCyFreeInstances` function returns one of the following codes:

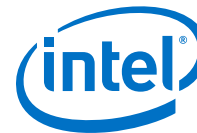
Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully freed the specified number of cy instances.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.

7.2.1.6 `icp_sal_userDcFreeInstances`

Free the specified number of data compression (dc) instances from the amount specified in the [DYN] section of the configuration file. The *numDcInstances* parameter specifies the number of dc instances to free.

Syntax

```
CpaStatus icp_sal_userDcFreeInstances ( Cpa32U numDcInstances,  
CpaInstanceHandle *pDcInstances );
```



Parameters

numDcInstances The number of dc instances to free.

***pDcInstances** A pointer to the dc instances to free.

Return Value

The `icp_sal_userDcInstancesAlloc` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully freed the specified number of dc instances.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.

7.2.1.7 `icp_sal_userCyGetAvailableNumDynInstancesByDevPkg`

Get the number of cryptographic instances that can be dynamically allocated using the `icp_sal_userCyGetAvailableNumDynInstancesByDevPkg` function.

Syntax

```
CpaStatus icp_sal_userCyGetAvailableNumDynInstancesByDevPkg (
Cpa32U *pNumCyInstances, Cpa32U devPkgID);
```

Parameters

***pNumCyInstances** A pointer to the number of cryptographic instances available for dynamic allocation.

devPkgID The device ID of the device of interest (Same as `accelID` in other APIs) If -1 then selects from all devices.

Return Value

The `icp_sal_userCyGetAvailableNumDynInstancesByDevPkg` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully retrieved the number of cryptographic instances available for dynamic allocation.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.



7.2.1.8 icp_sal_userDcGetAvailableNumDynInstancesByDevPkg

Get the number of data compression instances that can be dynamically allocated using the `icp_sal_userDcGetAvailableNumDynInstancesByDevPkg` function.

Syntax

```
CpaStatus icp_sal_userDcGetAvailableNumDynInstancesByDevPkg (
Cpa32U *pNumDcInstances, Cpa32U devPkgID);
```

Parameters

***pNumDcInstances** A pointer to the number of data compression instances available for dynamic allocation.

devPkgID The device ID of the device of interest (Same as `accelID` in other APIs) If -1 then selects from all devices.

Return Value

The `icp_sal_userDcGetAvailableNumDynInstancesByDevPkg` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully retrieved the number of cryptographic instances available for dynamic allocation.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.

7.2.1.9 icp_sal_userCyInstancesAllocByDevPkg

Allocate the specified number of cryptographic (cy) instances from the amount specified in the [DYN] section of the configuration file. The *numCyInstances* parameter specifies the number of cy instances to allocate and must be less than or equal to the value of the *NumberCyInstances* parameter in the [DYN] section of the configuration file.

Syntax

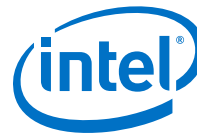
```
CpaStatus icp_sal_userCyInstancesAllocByDevPkg ( Cpa32U
numCyInstances, CpaInstanceHandle *pCyInstances, devPkgID);
```

Parameters

numCyInstances The number of cy instances to allocate.

***pCyInstances** A pointer to the cy instances.

devPkgID The device ID of the device of interest (Same as `accelID` in other APIs) If -1 then selects from all devices.



Return Value

The `icp_sal_userCyInstancesAllocByDevPkg` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully allocated the sepecified number of cy instances.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.

7.2.1.10 `icp_sal_userDcInstancesAllocByDevPkg`

Allocate the specified number of data compression (dc) instances from the amount specified in the [DYN] section of the configuration file. The *numDcInstances* parameter specifies the number of dc instances to allocate and must be less than or equal to the value of the *NumberDcInstances* parameter in the [DYN] section of the configuration file.

Syntax

```
CpaStatus icp_sal_userDcInstancesAllocByDevPkg ( Cpa32U
numDcInstances, CpaInstanceHandle *pDcInstances, devPkgID);
```

Parameters

`numDcInstances` The number of dc instances to allocate.

`*pDcInstances` A pointer to the dc instances.

`devPkgID` The device ID of the device of interest (Same as `accelID` in other APIs) If -1 then selects from all devices.

Return Value

The `icp_sal_userDcInstancesAllocByDevPkg` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully allocated the specified number of dc instances.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.



7.2.1.11 icp_sal_userCyGetAvailableNumDynInstancesByPkgAccel

Get the number of cryptographic instances that can be dynamically allocated using the `icp_sal_userCyGetAvailableNumDynInstancesByPkgAccel` function.

Syntax

```
CpaStatus icp_sal_userCyGetAvailableNumDynInstancesByPkgAccel (
Cpa32U *pNumCyInstances, Cpa32U devPkgID, Cpa32U
accelerator_number);
```

Parameters

***pNumCyInstances** A pointer to the number of cryptographic instances available for dynamic allocation.

devPkgID The device ID of the device of interest (Same as `accelID` in other APIs) If -1 then selects from all devices.

accelerator_number Accelerator Engine to use, valid values are 0..3

Return Value

The `icp_sal_userCyGetAvailableNumDynInstancesByPkgAccel` function returns one of the following codes:

Code

Meaning

`CPA_STATUS_SUCCESS` Successfully retrieved the number of cryptographic instances available for dynamic allocation.

`CPA_STATUS_FAIL` Indicates a failure.

7.2.1.12 icp_sal_userCyInstancesAllocByPkgAccel

Allocates the specified number of cryptographic (cy) instances from the amount specified in the [DYN] section of the configuration file. The *numCyInstances* parameter specifies the number of cy instances to allocate and must be less than or equal to the value of the *NumberCyInstances* parameter returned by a call to the `icp_sal_userCyInstancesAllocByPkgAccel` function.

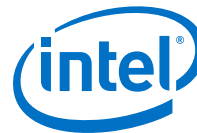
Syntax

```
CpaStatus icp_sal_userCyInstancesAllocByPkgAccel ( Cpa32U
numCyInstances, CpaInstanceHandle *pCyInstances, devPkgID, Cpa32U
accelerator_number);
```

Parameters

NumCyInstances The number of cy instances to allocate.

***pCyInstances** A pointer to the cy instances.



devPkgID The device ID of the device of interest (Same as accelID in other APIs) If -1 then selects from all devices.

accelerator_number Accelerator Engine to use, valid values are 0..3

Return Value

The `icp_sal_userCyInstancesAllocByDevPkg` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully allocated the specified number of cy instances.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.

7.2.2 IOMMU Remapping Functions

These functions are intended for IOMMU remapping operations.

All IOMMU remapping function definitions are located in: `$ICP_ROOT/quickassist/lookaside/access_layer/include/icp_sal_iommu.h`

The IOMMU remapping functions include:

- [icp_sal_iommu_get_remap_size](#) on page 115
- [icp_sal_iommu_map](#) on page 116
- [icp_sal_iommu_unmap](#) on page 116

7.2.2.1 icp_sal_iommu_get_remap_size

Returns the *page_size* rounded for IOMMU remapping.

Syntax

```
size_t icp_sal_iommu_get_remap_size (size_t size);
```

Parameters

size_t The minimum required page size.

Return Value

The `icp_sal_iommu_get_remap_size` function returns the *page_size* rounded for IOMMU remapping.



7.2.2.2 icp_sal_iommu_map

Adds an entry to the IOMMU remapping table.

Syntax

```
CpaStatus icp_sal_iommu_map ( Cpa64U phaddr, Cpa64U iova, size_t size);
```

Parameters

phaddr Host physical address.

iova Guest physical address.

size Size of the remapped region.

Return Value

The `icp_sal_iommu_map` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successful operation.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.

7.2.2.3 icp_sal_iommu_unmap

Removes an entry from the IOMMU remapping table.

Syntax

```
CpaStatus icp_sal_iommu_unmap ( Cpa64U iova, size_t size);
```

Parameters

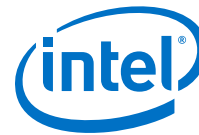
iova Guest physical address to be removed.

size Size of the remapped region.

Return Value

The `icp_sal_iommu_unmap` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successful operation.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.



7.2.2.4 IOMMU Remapping Function Usage

These functions are required when the user wants to access an acceleration service from the Physical Function (PF) when SR-IOV is enabled in the driver. In this case, all I/O transactions from the device go through DMA remapping hardware. This hardware checks 1) if the transaction is legitimate and 2) what physical address the given I/O address needs to be translated to. If the I/O address is not in the transaction table, it fails with a DMA Read error shown as follows:

```
DRHD: handling fault status reg 3
DMAR:[DMA Read] Request device [02:01.2] fault addr <ADDR>
DMAR:[fault reason 06] PTE Read access is not set
```

To make this work, the user must add a 1:1 mapping as follows:

1. Get the size required for a buffer:

```
int size = icp_sal_iommu_get_remap_size(size_of_data);
```

2. Allocate a buffer:

```
char *buff = malloc(size);
```

3. Get a physical pointer to the buffer:

```
buff_phys_addr = virt_to_phys(buff);
```

4. Add a 1:1 mapping to the IOMMU tables:

```
icp_sal_iommu_map(buff_phys_addr, buff_phys_addr, size);
```

5. Use the buffer to send data to the accelerator.
6. Before freeing the buffer, remove the IOMMU table entry:

```
icp_sal_iommu_unmap(buff_phys_addr, size);
```

7. Free the buffer:

```
free(buff);
```

The IOMMU remapping functions can be used in all contexts that the Intel® QuickAssist Technology APIs can be used, that is, kernel and user space in a Physical Function (PF) Dom0, as well as kernel and user space in a Virtual Machine (VM). In the case of VM, the APIs will do nothing. In the PF Dom0 case, the APIs will update the hardware IOMMU tables.

7.2.3 Polling Functions

These functions are intended for retrieving response messages that are on the rings and dispatching the associated callbacks.

All polling function definitions are located in: `$ICP_ROOT/quickassist/lookaside/access_layer/include/icp_sal_poll.h`



The polling functions include:

- [icp_sal_pollBank](#)
- [icp_sal_pollAllBanks](#)
- [icp_sal_CyPollInstance](#)
- [icp_sal_DcPollInstance](#)
- [icp_sal_CyPollDpInstance](#)
- [icp_sal_DcPollDpInstance](#)

7.2.3.1 [icp_sal_pollBank](#)

Poll all rings on the given accelerator on a given bank number to determine if any of the rings contain response messages from the Intel® QuickAssist Accelerator. The *response_quota* input parameter is per ring.

Syntax

```
CpaStatus icp_sal_pollBank ( Cpa32U accelId, Cpa32U bank_number,  
Cpa32U response_quota );
```

Parameters

accelId	The device number associated with the acceleration device. The valid range is 0 to the number of dh89xxcc devices in the system.
bank_number	The number of the memory bank on the dh89xxcc device that will be polled for response messages. The valid range is 0 to 7.
response_quota	The maximum number of responses to take from the ring in one call.

Return Value

The [icp_sal_pollBank](#) function returns one of the following codes:

Code	Meaning
<i>CPA_STATUS_SUCCESS</i>	Successfully polled a ring with data.
<i>CPA_STATUS_RETRY</i>	There is no data on any ring on any bank or the banks are already being polled.
<i>CPA_STATUS_FAIL</i>	Indicates a failure.



7.2.3.2 icp_sal_pollAllBanks

Poll all banks on the given acceleration device to determine if any of the rings contain response messages from the Intel® QuickAssist Accelerator. The *response_quota* input parameter is per ring.

Syntax

```
CpaStatus icp_sal_pollAllBanks ( Cpa32U accelId, Cpa32U
response_quota);
```

Parameters

accelId The device number associated with the acceleration device. The valid range is 0 to the number of dh89xxcc devices in the system.

response_quota The maximum number of responses to take from the ring in one call.

Return Value

The `icp_sal_pollAllBanks` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully polled a ring with data.
<code>CPA_STATUS_RETRY</code>	There is no data on any ring on any bank or the banks are already being polled.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.

7.2.3.3 icp_sal_CyPollInstance

Poll the cryptographic (Cy) logical instance associated with the *instanceHandle* to retrieve requests that are on response rings associated with that instance and dispatch the associated callbacks. The *response_quota* input parameter is the maximum number of responses to process in one call.

Note: The `icp_sal_CyPollInstance()` function is used in conjunction with the `CyXIsPolled` parameter in the acceleration configuration file. Refer to [Cryptographic Logical Instance Parameters](#) on page 156.

Syntax

```
CpaStatus icp_sal_CyPollInstance ( CpaInstanceHandle
instanceHandle, Cpa32U response_quota);
```

Parameters

instanceHandle The logical instance to poll for responses on the response ring.

response_quota The maximum number of responses to take from the ring in one call. When set to 0, all responses are retrieved.



Return Value

The `cp_sal_CyPollInstance` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	The function was successful.
<code>CPA_STATUS_RETRY</code>	There are no responses on the rings associated with the specified logical instance.

Note: A ring is only polled if it contains data.

<code>CPA_STATUS_FAIL</code>	Indicates a failure.
------------------------------	----------------------

7.2.3.4 `icp_sal_DcPollInstance`

Poll the data compression (Dc) logical instance associated with the *instanceHandle* to retrieve requests that are on response rings associated with that instance, and dispatch the associated callbacks. The *response_quota* input parameter is the maximum number of responses to process in one call.

Note: The `icp_sal_DcPollInstance()` function is used in conjunction with the `DcXIsPolled` parameter in the acceleration configuration file. Refer to [Data Compression Logical Instance Parameters](#) on page 157.

Syntax

```
CpaStatus icp_sal_DcPollInstance ( CpaInstanceHandle  
instanceHandle, Cpa32U response_quota );
```

Parameters

<code>instanceHandle</code>	The logical instance to poll for responses on the response ring.
<code>response_quota</code>	The maximum number of responses to take from the ring in one call. When set to 0, all responses are retrieved.

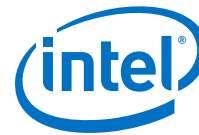
Return Value

The `icp_sal_DcPollInstance` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	The function was successful.
<code>CPA_STATUS_RETRY</code>	There are no responses on the rings associated with the specified logical instance.

Note: A ring is only polled if it contains data.

<code>CPA_STATUS_FAIL</code>	Indicates a failure.
------------------------------	----------------------



7.2.3.5 icp_sal_CyPollDpInstance

Poll a particular cryptographic (Cy) data path logical instance associated with the *instanceHandle* to retrieve requests that are on the high-priority symmetric ring associated with that instance and dispatch the associated callbacks. The *response_quota* input parameter is the maximum number of responses to process in one call.

Syntax

Note: This function is a Data Plane API function and consequently the restrictions in [Usage Constraints on the Data Plane APIs](#) on page 105 apply.

```
CpaStatus icp_sal_CyPollDpInstance ( CpaInstanceHandle
instanceHandle, Cpa32U response_quota);
```

Parameters

instanceHandle The logical instance to poll for responses on the response ring.

response_quota The maximum number of responses to take from the ring in one call. When set to 0, all responses are retrieved.

Return Value

The *icp_sal_CyPollDpInstance()* function returns one of the following codes:

Code	Meaning
<i>CPA_STATUS_SUCCESS</i>	The function was successful.
<i>CPA_STATUS_RETRY</i>	There are no responses on the rings associated with the specified logical instance.
<i>CPA_STATUS_FAIL</i>	Indicates a failure.

7.2.3.6 icp_sal_DcPollDpInstance

Poll a particular Data Compression (Dc) data path logical instance associated with the *instanceHandle* to retrieve requests that are on the response ring associated with that instance. The *response_quota* input parameter is the maximum number of responses to process in one call.

Syntax

Note: This function is a Data Plane API function and consequently the restrictions in [Usage Constraints on the Data Plane APIs](#) on page 105 apply.

```
CpaStatus icp_sal_DcPollDpInstance ( CpaInstanceHandle
instanceHandle, Cpa32U response_quota);
```



Parameters

- `instanceHandle` The logical instance to poll for responses on the response ring.
- `response_quota` The maximum number of responses to take from the ring in one call. When set to 0, all responses are retrieved.

Return Value

The `icp_sal_DcPollDpInstance` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	The function was successful.
<code>CPA_STATUS_RETRY</code>	There are no responses on the rings associated with the specified logical instance.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.

7.2.4 Random Number Generation Functions

These functions allow the configuration of the Intel® QuickAssist Technology random number generation APIs.

Non Deterministic Random Bit Generator (NRBG) Support

Also known as True Random Number Generator (TRNG), NRBG is only available on half of the crypto instances. Only two of the four Intel® Communications Chipset 8900 to 8920 Series device crypto accelerators offer TRNG hardware support. However, the user can employ the same entropy source with multiple DRBG instances.

In an Intel® Communications Chipset 8900 to 8920 Series device, the following accelerators are available:

- four Cipher slices
- four Authentication slices
- two TRNG

The NRBG slice can be accessed via the Intel® QuickAssist Technology NRBG API.

Deterministic Random Bit Generator (DRBG) Support

Implemented in software, DRBG processing takes some entropy as input and then performs Advanced Encryption Standard (AES) operations on the input using Intel® Communications Chipset 8900 to 8920 Series hardware.

The output is a deterministic random number. Once the user has the first random number from DRBG, the next number can be determined (assuming all AES parameters are known).

The DRBG in Intel® QuickAssist Technology is configured with an entropy source. One option is to use the Intel® QuickAssist Technology *NRBG* as the entropy source. This is what the performance sample code does but any other entropy source can also be configured (see the random number generation function list below).



All random number generation function definitions are located in the following header files:

- `$ICP_ROOT/quickassist/lookaside/access_layer/include/icp_sal_drbg_impl.h`
- `$ICP_ROOT/quickassist/lookaside/access_layer/include/icp_sal_drbg_ht.h`
- `$ICP_ROOT/quickassist/lookaside/access_layer/include/icp_sal_nrbg_ht.h`

The random number generation functions include:

- [icp_sal_drbgGetEntropyInputFuncRegister](#)
- [icp_sal_drbgGetInstance](#) on page 124
- [icp_sal_drbgGetNonceFuncRegister](#)
- [icp_sal_drbgHTGenerate](#)
- [icp_sal_drbgHTGetTestSessionSize](#)
- [icp_sal_drbgHTInstantiate](#)
- [icp_sal_drbgHTReseed](#)
- [icp_sal_drbgIsDFReqFuncRegister](#)
- [icp_sal_nrbgHealthTest](#)

The [icp_sal_drbgGetEntropyInputFuncRegister](#), [icp_sal_drbgGetNonceFuncRegister](#) or [icp_sal_drbgIsDFReqFuncRegister](#) functions must be called before calling any other Deterministic Random Bit Generator (DRBG) function.

The other functions should be called to validate that the DRBG is working correctly.

7.2.4.1 [icp_sal_drbgGetEntropyInputFuncRegister](#)

Allows the client to register a function that the implementation uses to retrieve inputs to the DRBG entropy source.

Syntax

```
IcpSalDrbgGetEntropyInputFunc  
icp_sal_drbgGetEntropyInputFuncRegister(  
IcpSalDrbgGetEntropyInputFunc func);
```

Parameters

func The function that the implementation may call to retrieve the DRBG entropy source.

Return Value

The [icp_sal_drbgGetEntropyInputFuncRegister](#) function returns the function that was previously registered with the implementation or NULL if no function was previously registered.



Sample Code

Refer to the sample application that demonstrates the random number generator capability provided by the software package in:

```
$ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/  
functional/sym/nrbg_sample/
```

7.2.4.2 icp_sal_drbgGetInstance

Retrieves the instance handle that DRBG is using.

Syntax

```
icp_sal_drbgGetInstance ( CpaCyDrbgSessionHandle sessionHandle,  
CpaInstanceHandle **pDrbgInstance);
```

Parameters

sessionHandle [in] The DRBG session handle structure that contains the session handle.

**pDrbgInstance [out] A pointer to the instance handle.

Return Value

None

7.2.4.3 icp_sal_drbgGetNonceFuncRegister

Allows the client to register a function that the implementation uses to retrieve the DRBG nonce.

Syntax

```
IcpSalDrbgGetNonceFunc icp_sal_drbgGetNonceFuncRegister (  
IcpSalDrbgGetNonceFunc func);
```

Parameters

func The function that the implementation may call to retrieve the nonce.

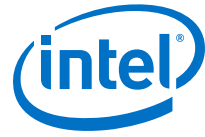
Return Value

The `icp_sal_drbgGetNonceFuncRegister` function returns the function that was previously registered with the implementation or NULL if no function was previously registered.

Sample Code

Refer to the sample application that demonstrates the random number generator capability provided by the software package in:

```
$ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/  
functional/sym/nrbg_sample/
```



7.2.4.4 icp_sal_drbgHTGenerate

Tests the health of the Generate function as described in NIST SP 800-90, section 11.3.3.

Syntax

```
CpaStatus icp_sal_drbgHTGenerate ( const CpaInstanceHandle
instanceHandle, IcpSalDrbgTestSessionHandle testSessionHandle);
```

Parameters

instanceHandle The handle of the instance for which DRBG is to be tested.

testSessionHandle The handle of the DRBG health test session. Physically contiguous memory for this session should be allocated by the client of the API.

Return Value

The `icp_sal_drbgHTGenerate` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Health tests passed.
<code>CPA_STATUS_FAIL</code>	Health tests failed.

7.2.4.5 icp_sal_drbgHTGetTestSessionSize

Gets the size of the contiguous memory that needs to be allocated by the user for the DRBG health test session.

Syntax

```
CpaStatus icp_sal_drbgHTGetTestSessionSize ( CpaInstanceHandle
instanceHandle, Cpa32U *pTestSessionSize);
```

Parameters

instanceHandle The handle of the instance for which DRBG is to be tested.

***pTestSessionSize** A pointer to a variable to store size of the memory required for DRBG health test session.

Return Value

The `icp_sal_drbgHTGetTestSessionSize` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully retrieved the health test session size.



Code	Meaning
<i>CPA_STATUS_FAIL</i>	Indicates a failure.

7.2.4.6 **icp_sal_drbgHTInstantiate**

Tests the health of Instantiate functionality as described in NIST SP 800-90, section 11.3.2. This function tests Instantiate for all possible setup configurations.

Syntax

```
CpaStatus icp_sal_drbgHTInstantiate ( const CpaInstanceHandle  
instanceHandle, IcpSalDrbgTestSessionHandle testSessionHandle );
```

Parameters

instanceHandle	The handle of the instance for which DRBG is to be tested.
testSessionHandle	The handle of the DRBG health test session. Physically contiguous memory for this session should be allocated by the client of the API.

Return Value

The `icp_sal_drbgHTInstantiate` function returns one of the following codes:

Code	Meaning
<i>CPA_STATUS_SUCCESS</i>	Health tests passed.
<i>CPA_STATUS_FAIL</i>	Health tests failed.

7.2.4.7 **icp_sal_drbgHTReseed**

Tests the health of the Reseed function as described in NIST SP 800-90, section 11.3.4.

Syntax

```
CpaStatus icp_sal_drbgHTReseed ( const CpaInstanceHandle  
instanceHandle, IcpSalDrbgTestSessionHandle testSessionHandle );
```

Parameters

instanceHandle	The handle of the instance for which DRBG is to be tested.
testSessionHandle	The handle of the DRBG health test session. Physically contiguous memory for this session should be allocated by the client of the API.

Return Value

The `icp_sal_drbgHTReseed` function returns one of the following codes:



Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Health tests passed.
<code>CPA_STATUS_FAIL</code>	Health tests failed.

7.2.4.8 `icp_sal_drbgIsDFReqFuncRegister`

Allows the client to register a function that the implementation uses to check if a derivation function is required.

Syntax

```
IcpSalDrbgIsDFReqFunc icp_sal_drbgIsDFReqFuncRegister(
IcpSalDrbgIsDFReqFunc func)
```

Parameters

func The function that the implementation may call to check if a derivation function is required.

Return Value

The `icp_sal_drbgIsDFReqFuncRegister` function returns the function that was previously registered with the implementation or NULL if no function was previously registered.

Sample Code

Refer to the sample application that demonstrates the random number generator capability provided by the software package in:

```
$ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/
functional/sym/nrbg_sample/
```

7.2.4.9 `icp_sal_nrbgHealthTest`

This function performs a check on the deterministic parts of the NRBG. It also provides the caller with the value of continuous random number generator test failures for $n=64$ bits. Refer to FIPS 140-2, section 4.9.2 for details. A non-zero value for the counter does not necessarily indicate a failure. It is statistically possible that consecutive blocks of 64 bits will be identical, and the RNG will discard the identical block in such cases. This counter allows the calling application to monitor changes in this counter and to use this to decide whether to mark the NRBG as faulty, based on the local policy or statistical model.

Syntax

```
CpaStatus icp_sal_nrbgHealthTest ( const CpaInstanceHandle
instanceHandle, Cpa32U *pContinuousRngTestFailures);
```

Parameters

instanceHandle The handle of the instance.



*pContinuousRngTestFailures The number of continuous random number generator test failures.

Return Value

The `icp_sal_nrbgHealthTest` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Health tests passed.
<code>CPA_STATUS_RETRY</code>	Resubmit the request.
<code>CPA_STATUS_INVALID_PARAM</code>	Invalid parameter passed in.
<code>CPA_STATUS_RESOURCE</code>	Error related to system resources.
<code>CPA_STATUS_FAIL</code>	Health tests failed.

Sample Code

Refer to the sample application that demonstrates the random number generator capability provided by the software package in:

```
$ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/  
functional/sym/nrbg_sample/
```

7.2.4.10 DRBG Health Test and `cpaCyDrbgSessionInit` Implementation Detail

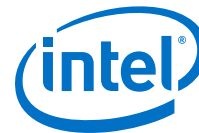
When using the acceleration driver for DRBG functionality, calls to `cpaCyDrbgSessionInit()` and the DRBG Health Test (DRBG HT) functions normally block while waiting for a response. Something (for example, another thread) is required to unblock the thread of execution.

When the application is using interrupts, this is not a problem. However, when the application is polling, this is a issue, especially for single-threaded applications, where there is no "polling thread".

Starting with software release 1.0.1, a build option has been added to the acceleration driver to allow the `cpaCyDrbgSessionInit(0)` and DRBG HT functions to poll for responses internally, rather than depending on an external polling thread. Instead of just waiting, these functions will now go into an internal loop, where they poll and wait with a pre-defined interval between polls (default 10 ms).

This functionality is automatically set at compile time in `user_space` only. It is not used in kernel space.

The default polling interval for `cpaCyDrbgSessionInit()` polling is 10 ms. This can be modified by adding the `drbgPollAndWaitTimeMS` parameter to the `GENERAL` section of the config file (see [General Parameters](#) on page 70). The polling in `cpaCyDrbgSessionInit()` is limited to the low-priority symmetric response ring to ensure that other rings in that instance do not have their responses polled.



Using the `DRBG_POLL_AND_WAIT` option at compile time now means that a polling application that needs to use the DRBG functionality can now be single-threaded and does not depend on a separate polling thread.

7.2.5 User Space Access Configuration Functions

Functions that allow the configuration of user space access to the Intel® QuickAssist Technology services from processes running in user space.

All user space access configuration function definitions are located in `$ICP_ROOT/quickassist/lookaside/access_layer/include/icp_sal_user.h`.

The user space access configuration functions include:

- `icp_sal_userStartMultiProcess`
- `icp_sal_userStart`
- `icp_sal_userStop`

7.2.5.1 `icp_sal_userStart`

Initializes user space access to an Intel® QuickAssist Accelerator and starts the services configured in the `pProcessName` section of the configuration file. This function needs to be called prior to any call to Intel® QuickAssist Technology API function from the user space process. This function is typically called only once in a user space process.

Note: The `icp_sal_userStart` function is for use only with the earlier configuration file variant (that is, the configuration file does **not** contain `ConfigVersion = 2`).

Syntax

```
CpaStatus icp_sal_userStart ( const char *pProcessName );
```

Parameters

***pProcessName** The name of the process corresponding to the section in the configuration file that defines and configures the services accessible to the process.

Return Value

The `icp_sal_userStart` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully started user space access to the Intel® QuickAssist Accelerator.
<code>CPA_STATUS_FAIL</code>	Operation failed.

Notes

None



7.2.5.2 icp_sal_userStartMultiProcess

Performs a function similar to `icp_sal_userStart()`, that is, initializes user space access to an Intel® QuickAssist Accelerator and starts the instances configured, if any, in the given section of the configuration file.

Note: The `icp_sal_userStartMultiProcess()` function is to be used with the simplified configuration file only (that is, the configuration file with `ConfigVersion = 2`).

The new configuration format allows the user to easily create a configuration for many user space processes. The driver internally generates unique process names and a valid configuration for each process based on the section name (`pSectionName`) and mode (`limitDevAccess`) provided.

For example, on an M device system, if all M configuration files contain:

```
[IPSec]
NumProcesses = N
LimitDevAccess = 0
```

then N internal sections are generated (each with instances on all devices) and N processes can be started at any given time. Each process can call `icp_sal_userStartMultiProcess("IPSec", CPA_FALSE)` and the driver determines the unique name to use for each process.

Similarly, on an M device system, if all M configuration files contain:

```
[SSL]
NumProcesses = N
LimitDevAccess=1
```

then M*N internal sections are generated (each with instances on one device only) and M*N processes can be started at any given time. Each process can call `icp_sal_userStartMultiProcess("SSL", CPA_TRUE)` and the driver determines the unique name to use for each process.

Refer to [Configuring Multiple Processes on a Multiple-Device System](#) on page 81 for a detailed example.

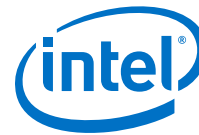
Syntax

```
CpaStatus icp_sal_userStartMultiProcess ( const char
*pSectionName, CpaBoolean limitDevAccess);
```

Parameters

***pSectionName** The section name described in the simplified configuration file format.

limitDevAccess Corresponds to the `LimitDevAccess` parameter setting in the simplified configuration file format.



Return Value

The `icp_sal_userStartMultiProcess` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successfully started user space access to the Intel® QuickAssist Accelerator as defined in the configuration file.
<code>CPA_STATUS_FAIL</code>	Operation failed.

7.2.5.2.1 `icp_sal_userStartMultiProcess` Usage

This topic describes a typical usage of the `icp_sal_userStartMultiProcess` function.

A common approach is as follows:

1. The user starts a main application (for example, an Apache web server or an OpenSSL speed application).
2. The main application spawns N child processes (workers). The number of child processes running at a given time should not be greater than the value configured by `NumProcesses` in the configuration file.
3. Each child process calls `icp_sal_userStartMultiProcess("SSL", CPA_TRUE)`. If the application spawns more child processes, the first N processes that call `icp_sal_userStartMultiProcess("SSL", CPA_TRUE)` start successfully with access to the accelerator. All subsequent calls start successfully but will not have access to the accelerator. In this case, calls to `cpaCyGetNumInstances()` and `cpaDcGetNumInstances()` return zero. If any of the N running processes finish their work and call `icp_sal_userStop()` (or if a subprocess terminates non-gracefully), another subprocess can call `icp_sal_userStartMultiProcess("SSL", CPA_TRUE)` and it will succeed.

7.2.5.3 `icp_sal_userStop`

Closes user space access to the Intel® QuickAssist Accelerator; stops the services that were running and frees the allocated resources. After a successful call to this function, user space access to the Intel® QuickAssist Accelerator from a calling process is not possible. This function should be called once when the process is finished using the Intel® QuickAssist Accelerator and does not intend to use it again.

Syntax

```
CpaStatus icp_sal_userStop ( void );
```

Parameters

None.

Return Value

The `icp_sal_userStop` function returns one of the following codes:



Code	Meaning
<i>CPA_STATUS_SUCCESS</i>	Successfully stopped user space access to the Intel® QuickAssist Accelerator.
<i>CPA_STATUS_FAIL</i>	Operation failed.

Notes

None

7.2.6 User Space Heartbeat Functions

These functions allow the user space application to check the status of the firmware/hardware of the Intel® Communications Chipset 8900 to 8920 Series device as part of the Heartbeat functionality.

All user space heartbeat function definitions are located in `$ICP_ROOT/quickassist/lookaside/access_layer/include/icp_sal_user.h`.

The heartbeat functions include:

- [icp_sal_check_device](#) on page 132
- [icp_sal_check_all_devices](#) on page 133

7.2.6.1 icp_sal_check_device

This function checks the status of the firmware/hardware for a given device and is used as part of the Heartbeat functionality.

Syntax

```
CpaStatus icp_sal_check_device ( Cpa32U accelID );
```

Parameters

accelID The device ID of the device of interest.

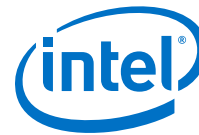
Return Value

The `icp_sal_check_device` function returns one of the following codes:

Code	Meaning
<i>CPA_STATUS_SUCCESS</i>	No error in operation.
<i>CPA_STATUS_FAIL</i>	Operation failed.

Notes

None



7.2.6.2 icp_sal_check_all_devices

This function checks the status of the firmware/hardware for all devices and is used as part of the Heartbeat functionality.

Syntax

```
CpaStatus icp_sal_check_all_devices ( void);
```

Parameters

None.

Return Value

The `icp_sal_check_all_devices` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	No error in operation.
<code>CPA_STATUS_FAIL</code>	Operation failed.

7.2.7 Version Information Function

A function that allows the retrieval of version information related to the software and hardware being used.

The version information function definition is located in: `$ICP_ROOT/quickassist/lookaside/access_layer/include/icp_sal_versions.h`.

There is only one version information function, that is, `icp_sal_getDevVersionInfo`.

7.2.7.1 icp_sal_getDevVersionInfo

Retrieves the hardware revision and information on the version of the software components being run on a given device.

Note: The `icp_sal_userStartMultiProcess` (or `icp_sal_userStart`) function must be called before calling this function. If not, calling this function returns `CPA_STATUS_INVALID_PARAM` indicating an error. The `icp_sal_userStartMultiProcess` (or `icp_sal_userStart`) function is responsible for setting up the ADF user space component, which is required for this function to operate successfully.

Syntax

```
CpaStatus icp_sal_getDevVersionInfo ( Cpa32U devId,  
icp_sal_dev_version_info_t *pVerInfo);
```

Parameters

`devId` The ID (number) of the device for which version information is to be retrieved.



*pVerInfo A pointer to a structure that holds the version information.

Return Value

The `icp_sal_getDevVersionInfo` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Operation finished successfully; version information retrieved.
<code>CPA_STATUS_INVALID_PARAM</code>	Invalid parameter passed to the function.
<code>CPA_STATUS_RESOURCE</code>	System resource problem.
<code>CPA_STATUS_FAIL</code>	Operation failed.

7.2.8 Reset Device Function

This API can only be called in user-space.

The device can be reset using this API call. This will schedule a reset of the device. See [Heartbeat Feature and Recovery from Hardware Errors](#) on page 46 for details of the steps on a device reset. The device can also be reset using the `adf_ctl` utility, e.g., by calling `adf_ctl icp_dev0 reset`.

7.2.8.1 `icp_sal_reset_device`

Resets the device.

Syntax

```
CpaStatus icp_sal_reset_device ( Cpa32U accelid );
```

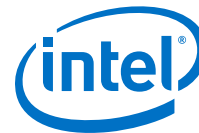
Parameters

`accelid` The device number.

Return Value

The `icp_sal_reset_device` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successful operation.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.



7.2.9 Thread-less APIs

These APIs can be used in the User Space Application when the driver is built with the ICP_WITHOUT_THREAD flag. See [Thread-less Mode](#) on page 53 for details.

The Thread-less API functions include:

- [icp_sal_poll_device_events](#) on page 135
- [icp_sal_find_new_devices](#) on page 135

7.2.9.1 icp_sal_poll_device_events

This reads any pending device events from icp_dev%d_csr (see [Driver Threading Model](#) on page 52) and forwards to interested subsystems.

Syntax

```
CpaStatus CpaStatus icp_sal_poll_device_events(void) (Cpa32U
accelid);
```

Parameters

none

Return Value

The `icp_sal_poll_device` function returns one of the following codes:

Code	Meaning
<code>CPA_STATUS_SUCCESS</code>	Successful operation.
<code>CPA_STATUS_FAIL</code>	Indicates a failure.

7.2.9.2 icp_sal_find_new_devices

This tries to connect to any available devices that the kernel driver has brought up and initialized for use in user space process.

Syntax

```
CpaStatus CpaStatus icp_sal_find_new_devices(void) (Cpa32U
accelid);
```

Parameters

none

Return Value

The `icp_sal_find_new_devices` function returns one of the following codes:



Code

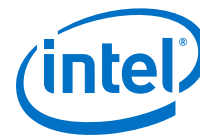
Meaning

CPA_STATUS_SUCCESS

Successful operation.

CPA_STATUS_FAIL

Indicates a failure.



Part 3: Applications and Usage Models



8.0 Application Usage Guidelines

This chapter provides some usage guidelines and identifies some of the applications to which the platforms described in this manual are ideally suited.

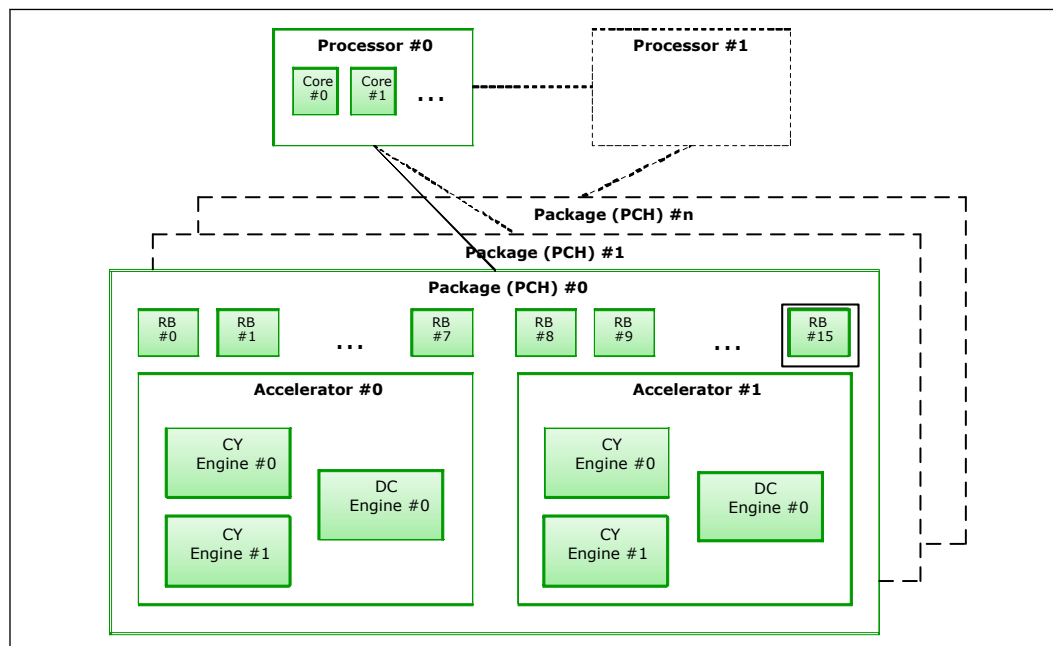
Note: The usage information provided in this section relates to the original configuration file format. Much of the information is still appropriate when using the newer (default) version of the configuration file.

8.1 Mapping Service Instances to Hardware Accelerators on the PCH

On the platform(s) described in this manual, a processor can be connected to one or more Intel® Communications Chipset 8900 to 8920 Series (PCH) devices. Each PCH device can contain zero, one or two accelerators depending on the device variant being used. An accelerator has one or more dedicated engines for each service type. Specifically, there are two cryptography engines and one data compression engine.

A set of 16 ring banks provide the communication mechanism between a processor and the acceleration complex on a PCH device. Each ring bank contains 16 individual rings for communication. The following figure shows the relationship between processors, PCH devices, accelerator(s) and ring banks.

Intel provides a driver as a starting point that abstracts the communication between the host and the rings and presents the high-level Intel® QuickAssist Technology APIs.

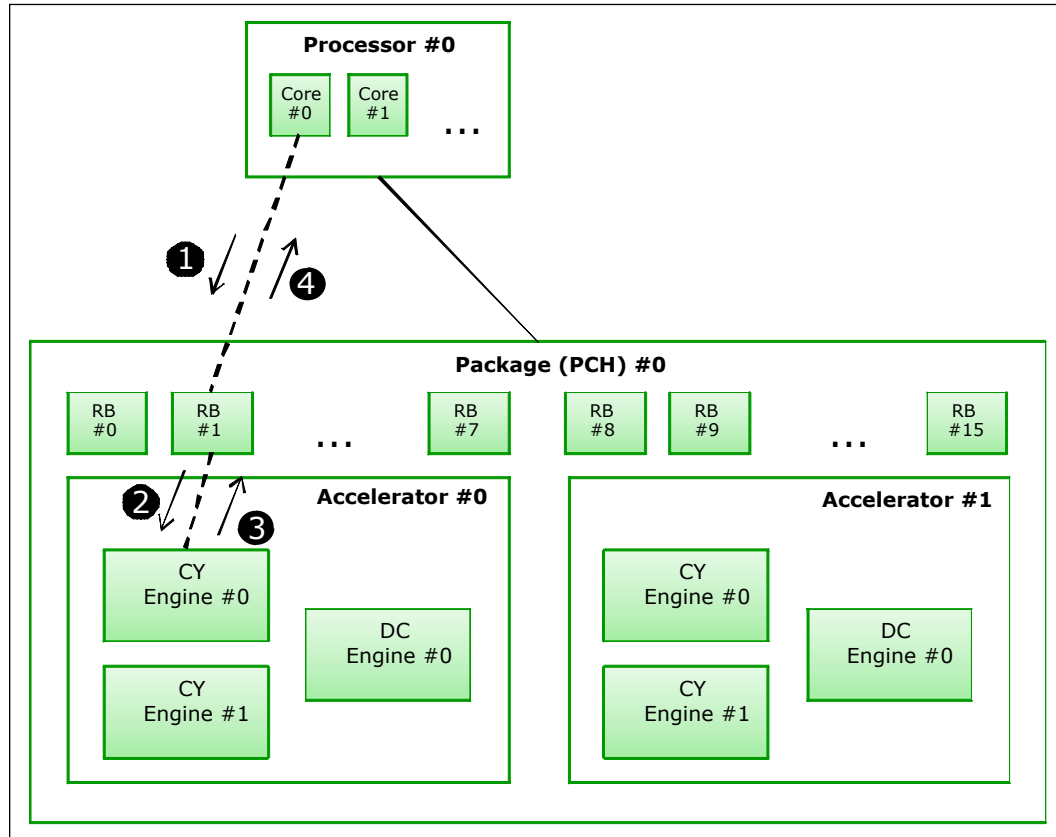
**Figure 24. Processor and PCH Device Components**

8.1.1 Processor and PCH Device Communication

An acceleration service uses different rings for request and response messages and for different priorities (currently for symmetric cryptography only). Communication between the processor and PCH device is achieved using the following operations (see also the following figure):

1. The processor uses a write (put) operation to place a request on the request ring.
2. The PCH device uses a read (get) operation to retrieve the request from the request ring.
3. Once the operation has been performed, the PCH device uses a write (put) operation to put the response to the response ring.
4. The processor uses a read (get) operation to retrieve the response from the response ring.

Figure 25. Processor and PCH Device Communication



8.1.2 Service Instances and Interaction with the Hardware

A typical use case would be to have a ring bank supporting two crypto instances and one compression instance.¹A service instance can be thought of as a channel between an accelerator and a core/thread running on the processor, which uses the rings for communication. The rings are not exposed by an API, but are set up using configuration files (one for each PCH device).

In general, a service instance uses a pair of rings, one for requests and one for responses. For cryptographic instances, separate request/response pairs are used for the following:

- Symmetric low priority
- Symmetric high priority
- Public key cryptography requests/responses

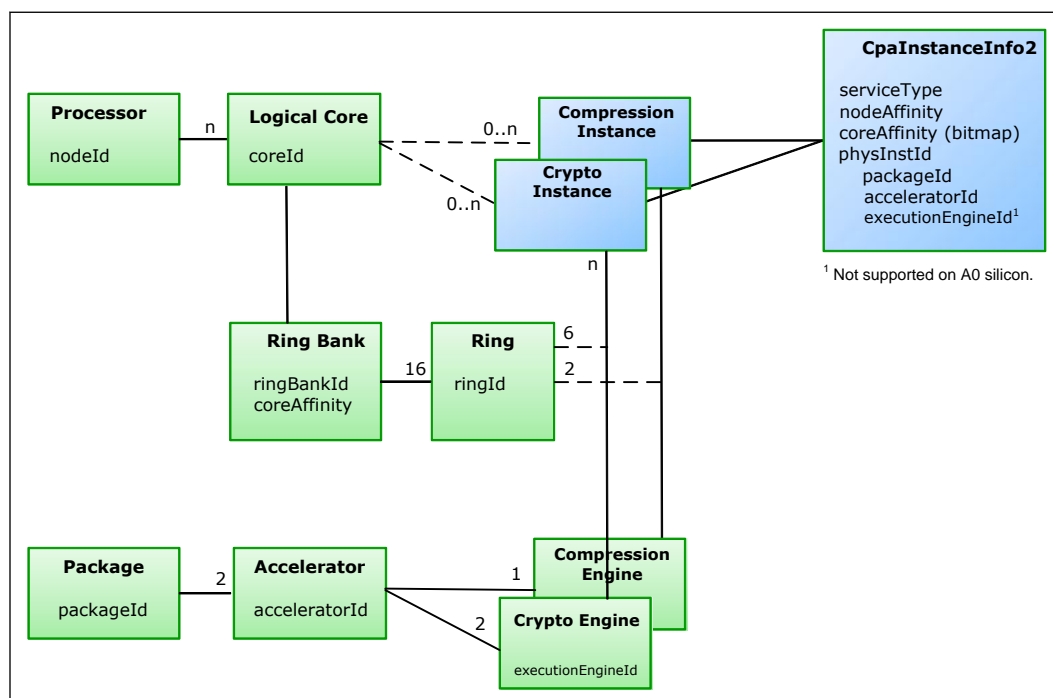
The key attributes of a service instance are given in the following table.

¹ The exceptions are ring bank 0, where two rings are reserved for administration related to accelerator 0 and ring bank 8, where two rings are reserved for administration related to accelerator 1. See [Figure 30](#) on page 150.

**Table 15. Service Instance Attributes**

Member	Sub-field	Description
physInstId	acceleratorId	Identifies the accelerator within the PCH
physInstId	executionEngineId	Identifies the engine (slice) within the accelerator
nodeAffinity	N/A	Identifies the processor node/socket to which the PCH is physically connected (relevant in NUMA configurations)
coreAffinity	N/A	Identifies the core(s) to which interrupts (if enabled) are affinitized (Bitmap)
isPolled	N/A	For Kernel space: <ul style="list-style-type: none"> IsPoll = 0 (interrupt mode) IsPoll = 1 (poll mode) For User space: <ul style="list-style-type: none"> IsPoll = 0 (interrupt mode) IsPoll = 1 (poll mode)

The following figure shows how the attributes relate to hardware components.

Figure 26. Service Instance Attributes and Hardware Components

8.1.3 Service Instance Configuration

The configuration of a service instance is done in the configuration file.

Note: The following example uses the earlier configuration file format, which continues to be supported.

The following figure shows an example extract of the relevant section in the configuration file.

Figure 27. Service Instance Configuration

```
#####
# User Space Instances Section
#####
[proc0] ❶
NumberCyInstances = 1
NumberDcInstances = 0

# Crypto - user space instance #0
Cy0Name = "proc0.0"
Cy0AcceleratorNumber = 0 ❷
Cy0ExecutionEngine = 0 ❸
Cy0BankNumber = 1 ❹
Cy0IsPolled = 1

Cy0RingAsymTx = 2 ❺
Cy0RingAsymRx = 3
Cy0RingSymTxHi = 4 ❻
Cy0RingSymTxLo = 5
Cy0RingSymRxHi = 6 ❼
Cy0RingSymRxLo = 7
```

In the previous figure, the meaning of each numbered item is explained as follows:

1. Each named address domain (one domain for the kernel, any number of user space process domains) has its own service instances.
2. Identifies the accelerator.
3. Identified the accelerator engine.
4. Identifies the ring bank to be used by the instances (which has a core affinity). See the configuration file snippet below for an example of core affinity association.
5. Asymmetric (public key), request (Tx) and response (Rx) rings.
6. Symmetric (bulk) crypto, low (normal) and high priority request (Tx) rings; *Cy0RingSymTxHi* is for the high priority requests and *Cy0RingSymTxLo* is for normal priority requests.
7. Symmetric (bulk) crypto, low (normal) and high priority response (Rx) rings; *Cy0RingSymRxHi* is for the high priority responses and *Cy0RingSymRxLo* is for normal priority responses.

Note: The data compression service requires just two rings; one for requests and one for responses.

8.1.4 Guidelines for Using Multiple Intel® QuickAssist Instances for Load Balancing in Cryptography Applications

The application is responsible for load balancing/spreading:

- Across engines within a PCH device
- Across PCH devices



To get the maximum performance from the hardware, there needs to be at least as many service instances as engines, that is:

- Four (two for each of the two accelerators in the top-end PCH SKU) for Cryptography.
- Two (per PCH device) for Data Compression.

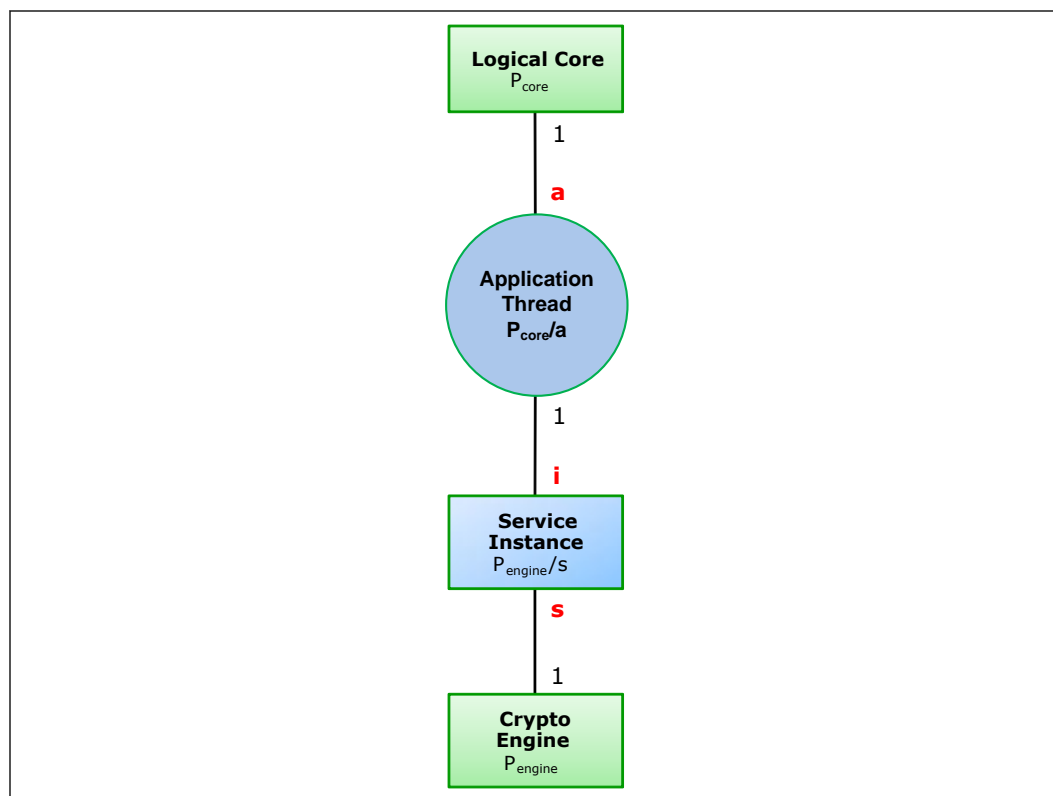
In the simplest case, load balancing is done through configuration. This applies when each engine has more capacity than required by an lcore. Each lcore uses exactly one service instance. Different lcores use different service instances, which map to different service engines. The load is balanced by spreading the traffic across lcores.

If a hardware engine has more capacity than that required by one instance, then multiple instances can share an engine. If a hardware engine has less capacity than required by one instance, then a core/process can talk to multiple instances.

Each core (physical or logical) has a certain application performance capacity (P_{core}). This depends on the core frequency, number of IA cycles per packet, packet size/mix, protocols and so on. Each (physical) service engine has a certain level of service performance (P_{engine}). This may depend on the PCH SKU, cryptography algorithms, packet size/mix and so on.

The following figure shows the relationship between cores, application threads, service instances and cryptographic engines.

Figure 28. Entities and Relationships for Load Balancing





The goal is to balance the performance of the cores and the service engines. Expressed mathematically, choose a , i and s (see figure), such that:

$$P_{\text{core}}/a \sim (P_{\text{engine}})*i/s$$

Note:

Performance capacity of a core may be measured as throughput at a certain packet size, mix of sizes, protocols and so on. Performance capacity of a service engine can be measured as throughput at a certain packet size, mix of sizes, algorithms and so on.

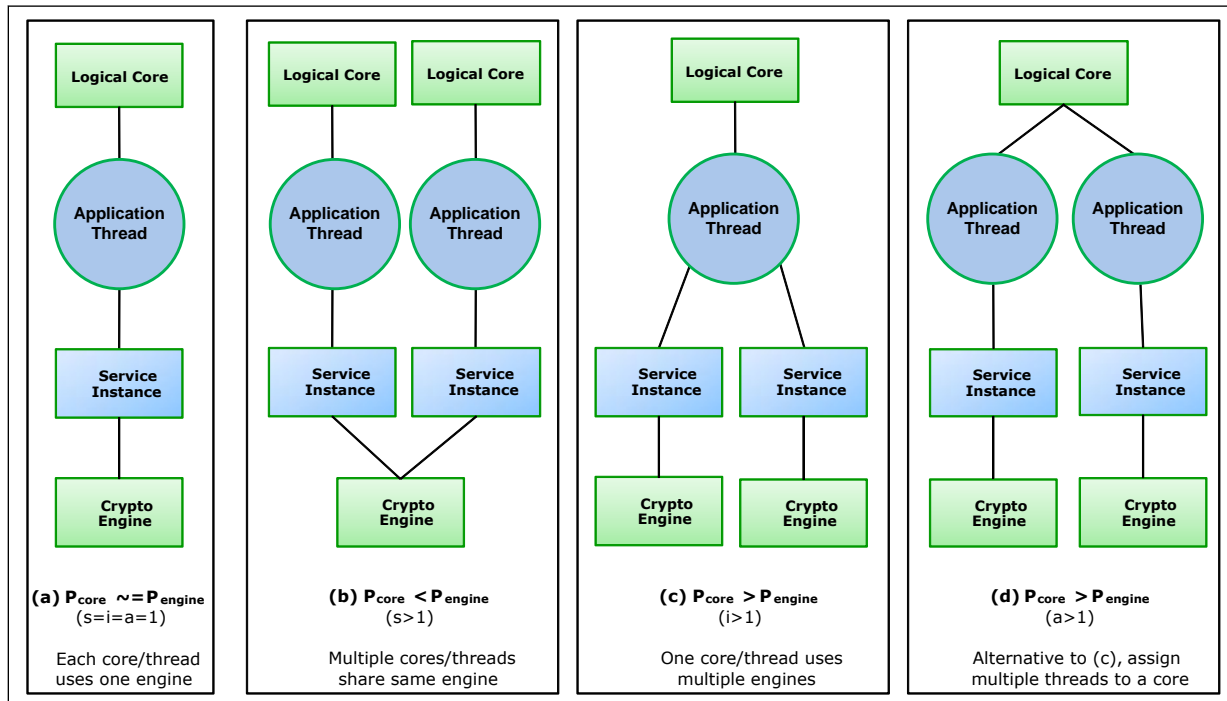
The following figure shows four different load balancing scenarios:

- Case (a) - The simplest case. Load balancing is done by spreading traffic across cores, and then each core talks to exactly one engine.
- Case (b) - When the engines have more capacity than the cores/threads need, then an engine can be mapped into multiple such cores/threads.

When the cores need more capacity than one engine can supply, then multiple engines must be mapped into a single core. There are at least two ways to do this as indicated in case (c) and (d) following.

- Case (c) - Each thread talks to multiple service instances. This requires the application code to change, in that the application must know about multiple instances and load balance across them.
- Case (d) - Multiple threads can be assigned to the same core. This moves the responsibility for load balancing to the OS or whatever is managing the threads.

Figure 29. Load Balancing Scenarios





In all cases, except Case (c), the code remains unchanged. Each thread talks to exactly one service instance. This makes it easier to port applications to different platforms with different numbers/frequencies of cores, different numbers/PCH SKU numbers and so on.

8.2 Cryptography Applications

Cryptography applications supported by the platforms described in this manual include, but are not limited to:

- Virtual Private Networks (VPNs, both IPsec and SSL). Both symmetric and public key cryptography can be offloaded for bulk transfer and key exchange (IKE, SSL handshakes and so on). See [IPsec and SSL VPNs](#) on page 145 for more information.
- Encrypted Storage. See [Encrypted Storage](#) on page 146 for more information.
- Web Proxy Appliances. See [Web Proxy Appliances](#) on page 146.

See also the [Accelerating a Security Appliance](#) white paper. This was first written to support the Intel® EP80579 Integrated Processor with Intel® QuickAssist Technology. Many of the concepts and ideas are applicable to the platforms described in this manual also.

8.2.1 IPsec and SSL VPNs

Virtual Private Networks (VPNs) allow for private networks to be established over the public internet by providing confidentiality, integrity and authentication using cryptography. VPN functionality can be provided by a standalone security gateway box at the boundary between the trusted and untrusted networks. It is also commonly combined with other networking and security functionality in a security appliance, or even in standard routers.

VPNs are typically based on one of two cryptographic protocols, either IPsec or DTLS. Each has its advantages and disadvantages.

One of the most compute-intensive aspects of a VPN is the cryptographic processing required to encrypt/decrypt traffic for confidentiality, to perform cryptographic hash functionality for authentication and to perform public key cryptography, based on modular exponentiation of large numbers or elliptic curve cryptography as part of key negotiation and exchange. The Intel® Communications Chipset 8900 to 8920 Series PCH provides cryptographic acceleration that can offload this computation from the CPU, thereby freeing up CPU cycles to perform other networking, security or other value-add applications.

The PCH offers its acceleration services through an API, called the Intel® QuickAssist Technology Cryptographic API. This can be invoked from the Linux* kernel or from Linux user space as well as from other operating systems. Intel also provides plugins to enable many of the PCH's cryptographic services to be accessed through open source cryptographic frameworks, such as the Linux kernel crypto framework/API (also known as the *scatterlist* API) and OpenSSL's libcrypto (through its EVP API). This facilitates ease of integration with certain open source implementations of protocol stacks, such as the Linux kernel's native IPsec stack (called NETKEY) or with OpenVPN (an open source SSL VPN implementation).



8.2.2 Encrypted Storage

In recent years, cases of lost laptops containing sensitive information have made the headlines all too frequently. Full disk encryption has become a standard procedure for many corporate PCs. Safe-guarding critical data however is not just a necessity in the client space, it is also a necessity in the data center.

Enterprise-class storage appliances achieve throughput rates in excess of 50 Gbps. Several high-profile cases of data theft have triggered updates to government regulations and industry standards. These regulations/standards now require protection of data-at-rest for applications involving sensitive data such as medical and financial records, typically using strong encryption. The high computational cost of adding security to storage appliances makes offload solutions an attractive value proposition.

Several complimentary standards for the security of data-at-rest exist, which when combined with traditional network security protocols, such as IPsec or SSL/TLS, provide an end-to-end secure storage solution, even for data-in-flight.

The IEEE Security in Storage working group is developing the IEEE 1619 series of standards that deal with cipher algorithms for disk and tape storage devices (AES in CCM and GCM modes). The cryptographic acceleration services of platforms that use the Intel® Communications Chipset 8900 to 8920 Series (PCH) are ideally suited for secure long-term storage solutions implementing the IEEE 1619.1 standard, by providing acceleration of the AES-256 cipher in CBC, CCM, and GCM modes and HMAC authentication using SHA-1, SHA-256 and SHA-512 hashes.

The Trusted Computing Group's (TCG) Storage Working Group does not prescribe a particular set of algorithms for the disk encryption. Instead, it defines several Storage Subsystem Classes (SSC) for various usage models, which define services such as enrollment and connection, protected storage (an extension of TPM), locking, logging, cryptographic services, authorization, and firmware updates. The cryptographic acceleration services of the platform can help by providing the highest level of security for authenticating the host to trusted peripherals implementing the TCG storage standards.

8.2.3 Web Proxy Appliances

Historically, Web Proxy appliances have evolved to present a public or intermediary interface for clients seeking resources from other servers, providing services such as web page caching and load balancing. These appliances are located at the edge of the network, typically at network gateways. Due to their centralized presence in the network, Web Proxy appliances today (referred to with a number of different names, such as Application Delivery Controllers, Reverse Proxy, and so on) have become a collection of services that include:

- Application Load Balancing (L4-L7)
- SSL Acceleration
- WAN Acceleration
- Caching
- Traffic Management
- Web Application Firewall



SSL and WAN acceleration have become common place capabilities of the Web Proxy appliance, requiring compute intensive algorithms for cryptography (SSL) and compression (WAN acceleration). Intel® Communications Chipset 8900 to 8920 Series (PCH) devices on the platforms described in this manual provide acceleration of asymmetric cryptography (RSA is the most commonly used key negotiation algorithm in SSL), symmetric cryptography (all algorithms defined in the TLS RFCs can be accelerated with the PCH) and compression (DEFLATE and LZS algorithms). With the prominence of Web Proxy appliances in typical networks, this use case has applications from cloud computing to small web server deployments.

8.3 Data Compression Applications

Data compression can be used as part of application delivery networks, data de-duplication, as well as in a number of crypto applications, for example, VPNs, IDS/IPS and so on.

8.3.1 Compression for Storage

In a time when the amount of online information is increasing dramatically, but budgets for storing that information remain static, compression technology is a powerful tool for improved information management, protection and access.

Compression appliances can transparently compress data such that clients can keep between two- and five-times more data online and reap the benefit of other efficiencies throughout the data lifecycle. By shrinking the primary data, all subsequent copies of that data, such as backups, archives, snapshots, and replicas are also compressed. Compression is the newest advancement in storage efficiency. Storage compression appliances can shrink primary online data in real time, without performance degradation. This can significantly lower storage capital and operating expenses by reducing the amount of data that is stored, and the required hardware that must be powered and cooled.

Compression can help slow the growth of storage, reducing storage costs while simplifying both operations and management. It also enables organizations to keep more data available for use, as opposed to storing data offsite or on harder-to-access media (such as tape).

Compression algorithms are very compute-intensive, which is one of the reasons why the adoption of compression techniques in mainstream applications has been slow. As an example, the DEFLATE Algorithm, which is one of the most used and popular compression techniques today, involves several compute-intensive steps: string search and match, sort logic, binary tree generation, Huffman Code generation. Intel® Communications Chipset 8900 to 8920 Series (PCH) devices in the platforms described in this manual provide acceleration capabilities in hardware that allow the CPU to offload the compute-intensive DEFLATE algorithm operations, thereby freeing up CPU cycles for other networking, security or other value-add operations.

8.3.2 Data Deduplication and WAN Acceleration

Data Deduplication and WAN Acceleration are coarse-grain data compression techniques centered around the concept of single-instance storage. Identical blocks of data (either to be stored on disk or to be transferred across a WAN link) are only stored/moved once, and any further occurrences are replaced by a reference to the first instance.



While the benefits of deduplication and WAN acceleration obviously depend on the type of data, multi-user collaborative environments are the most suitable due to the amount of naturally occurring replication caused by forwarded emails and multiple (similar) versions of documents in various stages of development.

Deduplication strategies can vary in terms of inline vs post-processing, block size granularity (file-level only, fixed block size or variable block-size chunking), duplicate identification (cryptographic hash only, simple CRC followed by byte-level comparison or hybrids) and duplicate look-up (for example, Bloom filter based index).

Cryptographic hashes are the most suitable techniques for reliably identifying matching blocks with an improbably low risk for false positives, but they also represent the most compute-intensive workload in the application. As such, the cryptographic acceleration services offered by the hardware (PCH) through the Intel® QuickAssist Technology Cryptographic API can be used to considerably improve the throughput of deduplication/WAN acceleration applications. Additionally, the compression/decompression acceleration services can be used to further compress blocks for storage on disk, while optionally encrypting the compressed contents for data security.



Appendix A Acceleration Driver Configuration File - Earlier File Format

Note: This chapter describes the older configuration file format. The older configuration file format is fully supported, but the format is deprecated in favor of the simpler new file format described earlier in this document.

This chapter describes the configuration file(s) managed by the Acceleration Driver Framework (ADF) that allow customization of runtime operation. This configuration file(s) must be tuned to meet the performance needs of the target application.

Note: The parameter values given in this chapter represent the configuration against which the software has been validated. While the configuration file is intended to be modified, no guarantee can be given for the expected behavior when parameter values are changed.

A.1 Configuration File Overview

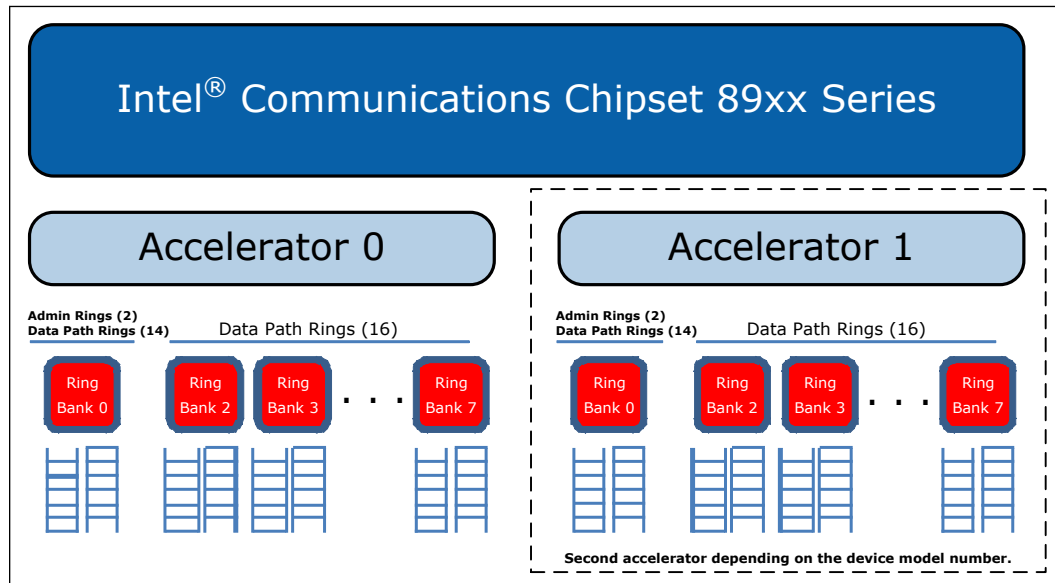
There is a single configuration file for each Intel® Communications Chipset 8900 to 8920 Series (PCH) device. The configuration file always contains two accelerator subsections. The significance of these subsections depends on the number of accelerators in the PCH device as defined by the model number:

- If there are no accelerators in the device, the information in both accelerator subsections is not relevant and can be ignored.
- If there is one accelerator in the device, only the information in the first accelerator subsection is relevant. The second subsection can be ignored.
- If there are two accelerators in the device, both accelerator subsections are relevant.

The client application may load balance between two accelerators if present.

Each accelerator has eight independent ring banks - the communication mechanism between the Acceleration software and the hardware. Each ring bank has an interrupt that can be directed to a specific Intel® architecture core. Each ring bank has 16 rings (hardware assisted queues). This hierarchy is shown in the following figure.

Figure 30. Ring Banks



Note: Depending on the SKU number, a PCH device may also contain no accelerators.

The configuration file is split into three (or more) sections: General, Hardware Access Ring Bank Configuration, and one or more Logical Instance sections.

- **General** - includes parameters that allow the user to:
 - Specify which services are enabled.
 - Configure the settings for the services.

Additional details are included in [General Parameters](#) on page 151.

- **Hardware Access Ring Bank Configuration** - includes parameters that allow the user to:
 - Enable and configure interrupt coalescing.
 - Direct an MSI-x interrupt for a given ring bank to a specified Intel® architecture core, assuming that the OS supports MSI-X interrupts.

Additional details are included in [\[AcceleratorX\] Section](#) on page 153.

- **Logical Instances** - one or more sections that include parameters that allow the user to:
 - Configure rings to be used by that address domain (kernel space or individual user space process) and define the behavior of the ring.

Additional details are included in [Logical Instances Section](#) on page 155.

A sample configuration file, targeted at a high-end IPsec box without compression, is included in [Sample Configuration File \(V1\)](#) on page 159.

A.2 General Section

The general section of the configuration file contains general parameters and statistics parameters.



A.2.1 General Parameters

The following table describes the parameters that can be included in the General section.

Please see [Table 6](#) on page 70

Table 16. General Parameters - Earlier File Format

Parameter	Description	Default	Range
ServicesEnabled	Defines the service(s) available (cryptographic [cyX], data compression [dc]).	cy0;dc	cyX, dc <i>Note:</i> X can be 0 or 1, which identifies one of two available cryptographic engines. <i>Note:</i> Multiple values permitted, use ; as the delimiter.
cyHmacAuthMode	Determines when HMAC precomputes are done.	1	1 - HMAC precomputes are done during session initialization 2 - HMAC precomputes are done during the perform operation <i>Note:</i> In general, with this parameter set to 1, performance is expected to be better.
dcTotalSRAMAvailable	Each PCH device has a total of 512 KB of eSRAM. The eSRAM can be used by different services, such as Data Compression. This parameter tells the driver how much of this memory to use for the Data Compression service. A value of 0 means, do not use any eSRAM for the Data Compression service; 512000 means use all the eSRAM for the Data Compression service.	0	0 to 512000 (currently, 0 is the only possible value, since eSRAM is not currently supported)
Firmware_MmpPath	Name of the Modular Math Processor (MMP) firmware.	mmp_firmware.bin	mmp_firmware.bin
<i>Note:</i> "Default" denotes the value in the configuration file when shipped.			

A.2.2 QAT Parameters

The following table describes accelerator-specific parameters.

Note: In the following parameters, beginning AccelX..., the X can be 0 or 1 representing the accelerator number.



Table 17. QAT Parameters - Earlier File Format

Parameter	Description	Default	Range
AccelXAdminBankNumber	Specifies the bank number for administration request/response rings on accelerator X, where X can be 0 or 1.	0	0 to 7
AccelXAcceleratorNumber	Specifies the accelerator number for administration request/response rings for accelerator X, where X can be 0 or 1.	0	0 or 1
AccelXAdminTx	Specifies the ring number of the administration request ring for accelerator X, where X can be 0 or 1.	0	0
AccelXAdminRx	Specifies the ring number of the administration response ring for accelerator X, where X can be 0 or 1.	1	1
<i>Note:</i> "Default" denotes the value in the configuration file when shipped.			

A.2.3 Statistics Parameters

The following table shows the parameters in the configuration file, prefixed with stats, that can be used to enable or disable certain types of statistics.

Note: There is a performance impact when statistics are enabled. In particular, the IA cost of offload is expected to increase when statistics are enabled.

When the statistics are enabled, the collected data can be retrieved using the following methods:

- Calling the appropriate Intel® QuickAssist Technology API function. For example, `cpaCySymQueryStats` or `cpaCySymQueryStats64` for symmetric cryptography. See the *Intel® QuickAssist Technology Cryptographic API Reference Manual* for more information about these functions.
- For kernel space instances, looking at entries in the `/proc/dh89xxcc_devX` directory, where X is the device number. For example, `/proc/icp_dh89xxcc_dev0/cy/IPSec0` for all statistics related to cryptography instance IPSec0, where IPSec0 is the name given to the instance in the config file (`Cy0Name = "IPSec0"`). See [Debug Feature](#) on page 44 for more information.

Table 18. Statistics Parameters

Parameter	Description	Default	Range
statsGeneral	Enables/disables statistics in general.	1	1 or 0
statsDc	Enables/disables statistics for data compression.	1	1 or 0
statsDh	Enables/disables statistics for the Diffie-Hellman algorithm.	1	1 or 0
statsDrbg	Enables/disables statistics for the Deterministic Random Bit Generator (DRBG).	1	1 or 0
statsDsa	Enables/disables statistics for the Digital Signature Algorithm (DSA).	1	1 or 0
<i>continued...</i>			



Parameter	Description	Default	Range
statsEcc	Enables/disables statistics for Elliptic Curve Cryptography (ECC).	1	1 or 0
statsKeyGen	Enables/disables statistics for the Key Generation algorithm.	1	1 or 0
statsLn	Enables/disables statistics for the Large Number generator.	1	1 or 0
statsPrime	Enables/disables statistics for the Prime Number detector.	1	1 or 0
statsRsa	Enables/disables statistics for the RSA algorithm.	1	1 or 0
statsSym	Enables/disables statistics for symmetric ciphers.	1	1 or 0
Note: "Default" denotes the value in the configuration file when shipped. A value of 1 indicates "enabled"; a value of 0 indicates "disabled".			

A.3 [AcceleratorX] Section

Note: A PCH device may contain 0, 1 or 2 accelerators depending on the model number. In the configuration file, there is an [AcceleratorX] section for each accelerator.

The [AcceleratorX] section of the configuration file contains interrupt coalescing and core affinity parameters.

A.3.1 Interrupt Coalescing Parameters

For each accelerator, the interrupt coalescing parameters in the following table can be configured.

Table 19. Interrupt Coalescing Parameters - Earlier File Format

Parameter	Description	Default	Range
BankXInterruptCoalescingEnabled	Specifies if interrupt coalescing is enabled for ring bank X, where X is in the range 0 to 7.	1	0 or 1
BankXInterruptCoalescingTimerNs	Specifies the coalescing time, in nanoseconds (ns), for ring bank X, where X is in the range 0 to 7. Note: If a value outside the range is set, the default value is used.	10000	500 to 1048575
BankXInterruptCoalescingNumResponses	Specifies the number of responses that need to arrive from hardware before the interrupt is triggered. It can be used to maximize throughput or adjust throughput latency ratio.	0 (disable)	0 to 248
Note: "Default" denotes the value in the configuration file when shipped.			

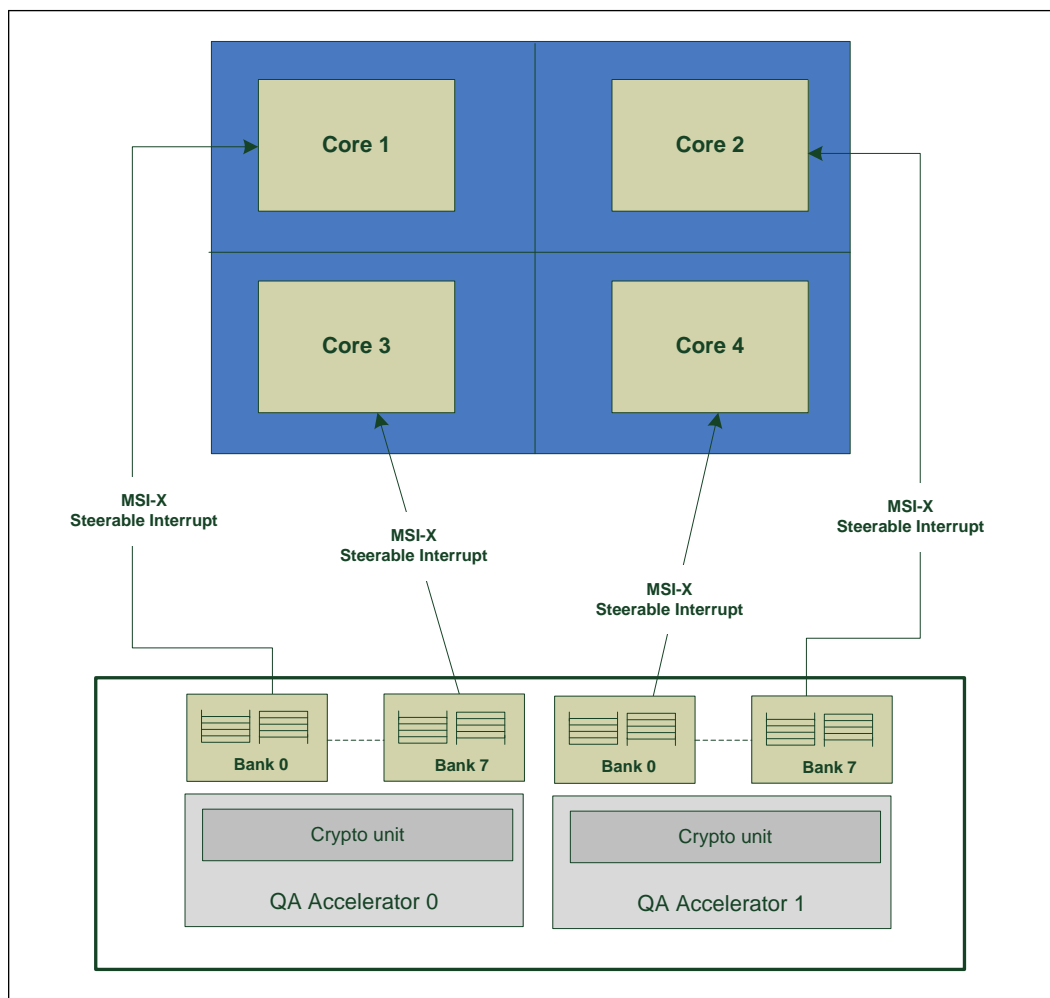
A.3.2 Affinity Parameters

To use core affinity, it is necessary to disable the `irqbalancer` service using the following command issued from an account with root privileges:

```
# service irqbalance stop
```

Each accelerator has eight ring banks (0 to 7). If the OS supports MSI-X interrupts, each ring bank has a steerable MSI-X interrupt that may be affinityized to a particular node/core as shown in the following figure.

Figure 31. Ring Bank Affinity to Core for MSI-X Interrupts



For each accelerator, the ring bank parameters in the following table can be configured.

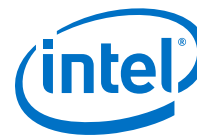


Table 20. Ring Bank Affinity Parameters

Parameter	Description	Default	Range
BankXCoreIDAffinity	Defines core affinity for ring bank X, where X is in the range 0 to 7.	0	0 to cpumax-1 <i>Note:</i> cpumax is the number of CPUs in the system.
<i>Note:</i> "Default" denotes the value in the configuration file when shipped.			

A.4 Logical Instances Section

A logical instance allows each address domain (kernel space and individual user space processes) to configure rings (hardware assisted queues) to be used by that address domain and to define the behavior of that ring. See [Hardware Assisted Rings](#) on page 28 and [Logical Instances](#) on page 23 for more information.

The address domains are in the following format:

- For the kernel address domain: [KERNEL]
- For user process address domains: [xxxxxx], where xxxxxx may be any ASCII value that uniquely identifies the user mode process.

To allow a driver to correctly configure the logical instances associated with this user process, the process must call the function [icp_sal_userStart](#) on page 129, passing the xxxxx string during process initialization. When the user space process is finished, it must call the function [icp_sal_userStop](#) on page 131 to free resources. See [User Space Access Configuration Functions](#) on page 129 for more information.

The items that can be configured for a logical instance are:

- The name of the logical instance
- The accelerator associated with this logical instance
- The ring bank associated with this logical instance
- The response mode associated with this logical instance (0 for IRQ, 1 for Polled)
- The ring for receiving and the ring for transmitting
- The number of concurrent requests supported by a pair of rings on this instance (Tx and Rx).

Note: This number affects the amount of memory allocated by the driver. Also, coalescing that is based on the number of responses is only enabled if: 1) Time-based coalescing is enabled, 2) The number of concurrent requests = 512256 (ring size = 16 KB) and 3) Bank<n>InterruptCoalescingNumResponses != 0.

Note: Logical instances may not share the same rings, but may share a ring bank.

A.4.1 [KERNEL] Section

In the [KERNEL] section of the configuration file, information about the number and type of kernel instances can be defined.

The following table describes the parameters that determine the number of kernel instances for each service.



Note: The maximum number of cryptographic instances supported is 32.

Parameter	Description	Default	Range
NumberCyInstances	Specifies the number of cryptographic instances. <i>Note:</i> Depends on the number of allocations to other services.	2	0 to 32
NumberDcInstances	Specifies the number of data compression instances. <i>Note:</i> Depends on the number of allocations to other services.	1	0 to 64
<i>Note:</i> "Default" denotes the value in the configuration file when shipped.			

A.4.1.1 Cryptographic Logical Instance Parameters

The following table shows the parameters that can be set for cryptographic logical instances.

Table 21. Cryptographic Logical Instance Parameters - Earlier File Format

Parameter	Description	Default	Range
CyXName	Specifies the name of cryptographic instance number X.	IPSec0	String (max. 64 characters)
CyXAcceleratorNumber	Specifies the accelerator number that the cryptographic instance number X is assigned to.	0	0 or 1
CyXBankNumber	Specifies the bank number of the cryptographic instance number X.	0 for kernel space instances 1 for user space instances	0 to 8
CyXExecutionEngine	Specifies the engine that cryptographic instance number X executes on.	0	0 or 1 (depending on the SKU)
CyXIsPolled	Specifies if cryptographic instance number X works in poll mode or IRQ mode.	0 for kernel space instances 1 for user space instances	For instance in the kernel space: 0 (interrupt mode) 1 (poll mode) For instance in the user space: 0 (interrupt mode) 1 (poll mode)
CyXNumConcurrentSymRequests	Specifies the number of cryptographic concurrent symmetric requests for cryptographic instance number X.	512	64, 128, 256, 512, 1024, 2048 or 4096
CyXNumConcurrentAsymRequests	Specifies the number of concurrent asymmetric requests for cryptographic instance number X.	64	64, 128, 256, 512, 1024, 2048 or 4096
CyXRingAsymTx	Specifies the asymmetric request ring number for cryptographic instance number X.	2 for kernel space instances	Even number in range: 0 to 14
continued...			



Parameter	Description	Default	Range
		0 for user space instances	
CyXRingAsymRx	Specifies the asymmetric response ring number for cryptographic instance number X.	3 for kernel space instances 1 for user space instances	Odd number in range: 1 to 15
	Specifies the symmetric request ring number for cryptographic instance number X for high priority messages.	4 for kernel space instances 2 for user space instances	Even number in range: 0 to 14
CyXRingSymTxLo	Specifies the symmetric request ring number for cryptographic instance number X for low priority messages.	5 for kernel space instances 3 for user space instances	Even number in range: 0 to 14
CyXRingSymRxHi	Specifies the symmetric response ring number for cryptographic instance number X for high priority messages.	6 for kernel space instances 4 for user space instances	Odd number in range: 0 to 15
CyXRingSymRxHi	Specifies the symmetric response ring number for cryptographic instance number X for low priority messages.	7 for kernel space instances 5 for user space instances	Odd number in range: 1 to 15

Note: "Default" denotes the value in the configuration file when shipped.

A.4.1.2 Data Compression Logical Instance Parameters

The following table shows the parameters in the configuration file that can be set for data compression logical instances.

Note: The maximum number of data compression instances supported is 126.

Parameter	Description	Default	Range
DcXName	Specifies the name of data compression instance number X.	IPComp0	String (max. 64 characters)
DcXAcceleratorNumber	Specifies the accelerator number that the data compression instance number X is assigned to.	0	0 or 1
DcXBankNumber	Specifies the bank number of data compression instance number X.	0 for kernel space instances 1 for user space instances	0 to 8

continued...



Parameter	Description	Default	Range
DcXIsPolled	Specifies if data compression instance number X works in poll mode or IRQ mode.	0 for kernel space instances 1 for user space instances	For instance in the kernel space: 0 (interrupt mode) 1 (poll mode) For instance in the user space: 0 (interrupt mode) 1 (poll mode)
DcXNumConcurrentRequests	Specifies the number of data compression concurrent requests.	512	64, 128, 256, 512, 1024, 2048 or 4096
DcXRingTx	Specifies the request ring number for data compression instance number X.	8 for kernel space instances 6 for user space instances	Even number in the range: 0 to 14
DcXRingRx	Specifies the response ring number for data compression instance number X.	9 for kernel space instances 7 for user space instances	Odd number in the range: 1 to 15
Note: "Default" denotes the value in the configuration file when shipped.			

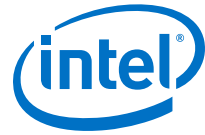
A.4.2 User Process Instance [xxxxx] Sections

In each [xxxxx] section of the configuration file, information about the number and type of user process instances can be defined.

The parameters in the following table specify the number of user process instances for each service.

Parameter	Description	Default	Range
NumberCyInstances	Specifies the number of cryptographic instances. Note: Depends on the number of allocations to other services.	0	0 to 32
NumberDcInstances	Specifies the number of data compression instances. Note: Depends on the number of allocations to other services.	0	0 to 126
Note: "Default" denotes the value in the configuration file when shipped.			

Parameters for each user process instance can also be defined. The parameters that can be included for each specific user process instance are similar to those in the [Logical Instances Section](#) on page 155.



A.5 Sample Configuration File (V1)

The following sample configuration file is intended for a high-end IPsec box.

```
#####
#
# @par
# This file is provided under a dual BSD/GPLv2 license.  When using or
# redistributing this file, you may do so under either license.
#
#   GPL LICENSE SUMMARY
#
#   Copyright(c) 2007-2013 Intel Corporation. All rights reserved.
#
#   This program is free software; you can redistribute it and/or modify
#   it under the terms of version 2 of the GNU General Public License as
#   published by the Free Software Foundation.
#
#   This program is distributed in the hope that it will be useful, but
#   WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
#   General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#   Foundation, Inc., 51 Franklin St - Fifth Floor, Boston, MA 02110-1301 USA.
#   The full GNU General Public License is included in this distribution
#   in the file called LICENSE.GPL.
#
#   Contact Information:
#   Intel Corporation
#
#   BSD LICENSE
#
#   Copyright(c) 2007-2013 Intel Corporation. All rights reserved.
#   All rights reserved.
#
#   Redistribution and use in source and binary forms, with or without
#   modification, are permitted provided that the following conditions
#   are met:
#
#       * Redistributions of source code must retain the above copyright
#       notice, this list of conditions and the following disclaimer.
#       * Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in
#       the documentation and/or other materials provided with the
#       distribution.
#       * Neither the name of Intel Corporation nor the names of its
#       contributors may be used to endorse or promote products derived
#       from this software without specific prior written permission.
#
#   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
#   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
#   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
#   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
#   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
#   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
#   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
#   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
#   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
#   version: QAT1.5.L.1.10.0-65
#####
# This file is the configuration for a single dh89xxcc_qa
```



```
# device.
#
# Each device has up to two accelerators.
# - The client may load balance between these
#   accelerators.
# Each accelerator has 8 independent ring banks.
# - The interrupt for each can be directed to a
#   specific core.
# Each ring bank as 16 rings (hardware assisted queues).
#
#####

#####
# General Section
#####

[GENERAL]
#ServicesEnabled = cy0;cy1;dc
ServicesEnabled = cy0;cy1;dc

# Look Aside Cryptographic Configuration
cyHmacAuthMode = 1

# Look Aside Compression Configuration
dcTotalSRAMAvailable = 0

#No wireless
NumberOfWirelessProcs = 0

# Firmware Location Configuration
Firmware_MofPath = mof_firmware.bin
Firmware_MmpPath = mmp_firmware.bin

# QAT Parameters
Accel0AdminBankNumber = 0
Accel0AcceleratorNumber = 0
Accel0AdminTx = 0
Accel0AdminRx = 1

Accel1AcceleratorNumber = 1
Accel1AdminBankNumber = 0
Accel1AdminTx = 0
Accel1AdminRx = 1

#Statistics, valid values: 1,0
statsGeneral = 1
statsDc = 1
statsDh = 1
statsDrbg = 1
statsDsa = 1
statsEcc = 1
statsKeyGen = 1
statsLn = 1
statsPrime = 1
statsRsa = 1
statsSym = 1

# Enables or disables Single Root Complex IO Virtualization.
# If this is enabled (1) then SRIOV and VT-d need to be enabled in
# BIOS and there can be no Cy or Dc instances created in PF (Dom0).
# If this is disabled (0) then SRIOV and VT-d need to be disabled
# in BIOS and Cy and/or Dc instances can be used in PF (Dom0)
SRIOV_Enabled = 0

#Debug feature, if set to 1 it enables additional entries in /proc filesystem
ProcDebug = 1

#####
#
# Hardware Access Ring Bank Configuration
# Each Accelerator has 8 ring banks (0-7)
```




```
# If the OS supports MSI-X, each ring bank has an
# steerable MSI-x interrupt which may be
# affinitized to a particular node/core.
#
#####
```

```
[Accelerator0]
Bank0InterruptCoalescingEnabled = 1
Bank0InterruptCoalescingTimerNs = 10000
Bank0CoreIDAffinity = 0
Bank0InterruptCoalescingNumResponses = 0
```

```
Bank1InterruptCoalescingEnabled = 1
Bank1InterruptCoalescingTimerNs = 10000
Bank1CoreIDAffinity = 1
Bank1InterruptCoalescingNumResponses = 0
```

```
Bank2InterruptCoalescingEnabled = 1
Bank2InterruptCoalescingTimerNs = 10000
Bank2CoreIDAffinity = 0
Bank2InterruptCoalescingNumResponses = 0
```

```
Bank3InterruptCoalescingEnabled = 1
Bank3InterruptCoalescingTimerNs = 10000
Bank3CoreIDAffinity = 1
Bank3InterruptCoalescingNumResponses = 0
```

```
Bank4InterruptCoalescingEnabled = 1
Bank4InterruptCoalescingTimerNs = 10000
Bank4CoreIDAffinity = 0
Bank4InterruptCoalescingNumResponses = 0
```

```
Bank5InterruptCoalescingEnabled = 1
Bank5InterruptCoalescingTimerNs = 10000
Bank5CoreIDAffinity = 2
Bank5InterruptCoalescingNumResponses = 0
```

```
Bank6InterruptCoalescingEnabled = 1
Bank6InterruptCoalescingTimerNs = 10000
Bank6CoreIDAffinity = 4
Bank6InterruptCoalescingNumResponses = 0
```

```
Bank7InterruptCoalescingEnabled = 1
Bank7InterruptCoalescingTimerNs = 10000
Bank7CoreIDAffinity = 6
Bank7InterruptCoalescingNumResponses = 0
```

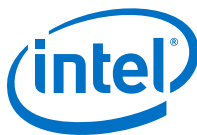
```
[Accelerator1]
Bank0InterruptCoalescingEnabled = 1
Bank0InterruptCoalescingTimerNs = 10000
Bank0CoreIDAffinity = 2
Bank0InterruptCoalescingNumResponses = 0
```

```
Bank1InterruptCoalescingEnabled = 1
Bank1InterruptCoalescingTimerNs = 10000
Bank1CoreIDAffinity = 3
Bank1InterruptCoalescingNumResponses = 0
```

```
Bank2InterruptCoalescingEnabled = 1
Bank2InterruptCoalescingTimerNs = 10000
Bank2CoreIDAffinity = 1
Bank2InterruptCoalescingNumResponses = 0
```

```
Bank3InterruptCoalescingEnabled = 1
Bank3InterruptCoalescingTimerNs = 10000
Bank3CoreIDAffinity = 0
Bank3InterruptCoalescingNumResponses = 0
```

```
Bank4InterruptCoalescingEnabled = 1
Bank4InterruptCoalescingTimerNs = 10000
```



```
Bank4CoreIDAffinity = 1
Bank4InterruptCoalescingNumResponses = 0

Bank5InterruptCoalescingEnabled = 1
Bank5InterruptCoalescingTimerNs = 10000
Bank5CoreIDAffinity = 3
Bank5InterruptCoalescingNumResponses = 0

Bank6InterruptCoalescingEnabled = 1
Bank6InterruptCoalescingTimerNs = 10000
Bank6CoreIDAffinity = 5
Bank6InterruptCoalescingNumResponses = 0

Bank7InterruptCoalescingEnabled = 1
Bank7InterruptCoalescingTimerNs = 10000
Bank7CoreIDAffinity = 7
Bank7InterruptCoalescingNumResponses = 0

#####
#
# Logical Instances Section
# A logical instance allows each address domain
# (kernel space and individual user space processes)
# to configure rings (i.e. hardware assisted queues)
# to be used by that address domain and to define the
# behavior of that ring.
#
# The address domains are in the following format
# - For kernel address domains
#   [KERNEL]
# - For user process address domains
#   [xxxxx]
# Where xxxxx may be any ascii value which uniquely identifies
# the user mode process.
# To allow the driver correctly configure the
# logical instances associated with this user process,
# the process must call the icp_sal_userStart(...)
# passing the xxxxx string during process initialisation.
# When the user space process is finish it must call
# icp_sal_userStop(...) to free resources.
# If there are multiple devices present in the system all conf
# files that describe the devices must have the same address domain
# sections even if the address domain does not configure any instances
# on that particular device. So if icp_sal_userStart("xxxxx") is called
# then user process address domain [xxxxx] needs to be present in all
# conf files for all devices in the system.
#
# Items configurable by a logical instance are:
# - Name of the logical instance
# - The accelerator associated with this logical
#   instance
# - The execution engine associated with this logical
#   instance (For crypto instances only)
# - The ring bank associated with this logical
#   instance.
# - The response mode associated with this logical instance (0
#   for IRQ or 1 for polled).
# - The ring for receiving and the ring for transmitting.
# - The number of concurrent requests supported by a pair of
#   rings on this instance (tx + rx). Note this number affects
#   the amount of memory allocated by the driver. Also
#   Bank<n>InterruptCoalescingNumResponses is only supported for
#   number of concurrent requests equal to 512.
#
# Note: Logical instances may not share the same ring, but
#       may share a ring bank.
#
# The format of the logical instances are:
# - For crypto:
#       Cy<n>Name = "xxxx"
#       Cy<n>AcceleratorNumber = 0|1
```



```
#           Cy<n>ExecutionEngine = 0|1
#           Cy<n>BankNumber = 0-7
#           Cy<n>IsPolled = 0|1
#           Cy<n>NumConcurrentSymRequests = 64|128|256|512|1024|2048|4096
#           Cy<n>NumConcurrentAsymRequests = 64|128|256|512|1024|2048|4096
#           Cy<n>RingAsymTx = 0-14 (Even numbers only)
#           Cy<n>RingAsymRx = 1-15 (Odd numbers only)
#           Cy<n>RingSymTxHi = 0-14 (Even numbers only)
#           Cy<n>RingSymRxHi = 1-15 (Odd numbers only)
#           Cy<n>RingSymTxLo = 0-14 (Even numbers only)
#           Cy<n>RingSymRxLo = 1-15 (Odd numbers only)
# Note:
#       The value Cy<n>NumConcurrentAsymRequests will do impact to memory
#       consumption greatly. Below is some memory consumption data for
#       the configuration per instance.
#       128: 10M
#       512: 40M
#       1024: 78M
#       4096: 280M
#       By default, 4 kernel instances and 4 user space instances, so if
#       the value is set to be 4096, for pke, the memory consumption is:
#       (4+4)*280=2240M
#
# - For Data Compression
#       Dc<n>Name = "xxxx"
#       Dc<n>AcceleratorNumber = 0|1
#       Dc<n>BankNumber = 0-7
#       Dc<n>IsPolled = 0|1
#       Dc<n>NumConcurrentRequests = 64|128|256|512|1024|2048|4096
#       Dc<n>RingTx = 0-14 (Even numbers only)
#       Dc<n>RingRx = 1-15 (Odd numbers only)
#
# Where:
#       - n is the number of this logical instance starting at 0.
#       - xxxx may be any ascii value which identifies the logical instance.
#
#####

#####
# Kernel Instances Section
#####
[KERNEL]
NumberCyInstances = 4
NumberDcInstances = 2

# Crypto - Kernel instance #0
Cy0Name = "IPSec0"
Cy0AcceleratorNumber = 0
Cy0ExecutionEngine = 0
Cy0BankNumber = 0
Cy0IsPolled = 0
Cy0NumConcurrentSymRequests = 512
Cy0NumConcurrentAsymRequests = 128
Cy0RingAsymTx = 2
Cy0RingAsymRx = 3
Cy0RingSymTxHi = 4
Cy0RingSymRxHi = 5
Cy0RingSymTxLo = 6
Cy0RingSymRxLo = 7

# Crypto - Kernel instance #1
CylName = "IPSec1"
CylAcceleratorNumber = 0
CylExecutionEngine = 1
CylBankNumber = 1
CylIsPolled = 0
CylNumConcurrentSymRequests = 512
CylNumConcurrentAsymRequests = 128
CylRingAsymTx = 0
CylRingAsymRx = 1
```



```
CylRingSymTxHi = 2
CylRingSymRxHi = 3
CylRingSymTxLo = 4
CylRingSymRxLo = 5

# Crypto - Kernel instance #2
Cy2Name = "IPSec2"
Cy2AcceleratorNumber = 1
Cy2ExecutionEngine = 0
Cy2BankNumber = 0
Cy2IsPolled = 0
Cy2NumConcurrentSymRequests = 512
Cy2NumConcurrentAsymRequests = 128
Cy2RingAsymTx = 2
Cy2RingAsymRx = 3
Cy2RingSymTxHi = 4
Cy2RingSymRxHi = 5
Cy2RingSymTxLo = 6
Cy2RingSymRxLo = 7

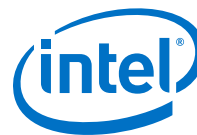
# Crypto - Kernel instance #3
Cy3Name = "IPSec3"
Cy3AcceleratorNumber = 1
Cy3ExecutionEngine = 1
Cy3BankNumber = 1
Cy3IsPolled = 0
Cy3NumConcurrentSymRequests = 512
Cy3NumConcurrentAsymRequests = 128
Cy3RingAsymTx = 0
Cy3RingAsymRx = 1
Cy3RingSymTxHi = 2
Cy3RingSymRxHi = 3
Cy3RingSymTxLo = 4
Cy3RingSymRxLo = 5

# Data Compression - Kernel instance #0
Dc0Name = "IPComp0"
Dc0AcceleratorNumber = 0
Dc0BankNumber = 0
Dc0IsPolled = 0
Dc0NumConcurrentRequests = 512
Dc0RingTx = 8
Dc0RingRx = 9

# Data Compression - Kernel instance #1
Dc1Name = "IPComp1"
Dc1AcceleratorNumber = 1
Dc1BankNumber = 2
Dc1IsPolled = 0
Dc1NumConcurrentRequests = 512
Dc1RingTx = 0
Dc1RingRx = 1

#####
# User Process Instance Section
#####
[SSL]
NumberCyInstances = 4
NumberDcInstances = 2

# Crypto - User instance #0
Cy0Name = "SSL0"
Cy0AcceleratorNumber = 0
Cy0ExecutionEngine = 0
Cy0BankNumber = 0
Cy0IsPolled = 1
Cy0NumConcurrentSymRequests = 512
Cy0NumConcurrentAsymRequests = 128
Cy0RingAsymTx = 10
Cy0RingAsymRx = 11
Cy0RingSymTxHi = 12
```



```
Cy0RingSymRxHi = 13
Cy0RingSymTxLo = 14
Cy0RingSymRxLo = 15

# Crypto - User instance #1
CylName = "SSL1"
CylAcceleratorNumber = 0
CylExecutionEngine = 1
CylBankNumber = 1
CylIsPolled = 1
CylNumConcurrentSymRequests = 512
CylNumConcurrentAsymRequests = 128
CylRingAsymTx = 6
CylRingAsymRx = 7
CylRingSymTxHi = 8
CylRingSymRxHi = 9
CylRingSymTxLo = 10
CylRingSymRxLo = 11

# Crypto - User instance #2
Cy2Name = "SSL2"
Cy2AcceleratorNumber = 1
Cy2ExecutionEngine = 0
Cy2BankNumber = 0
Cy2IsPolled = 1
Cy2NumConcurrentSymRequests = 512
Cy2NumConcurrentAsymRequests = 128
Cy2RingAsymTx = 8
Cy2RingAsymRx = 9
Cy2RingSymTxHi = 10
Cy2RingSymRxHi = 11
Cy2RingSymTxLo = 12
Cy2RingSymRxLo = 13

# Crypto - User instance #3
Cy3Name = "SSL3"
Cy3AcceleratorNumber = 1
Cy3ExecutionEngine = 1
Cy3BankNumber = 1
Cy3IsPolled = 1
Cy3NumConcurrentSymRequests = 512
Cy3NumConcurrentAsymRequests = 128
Cy3RingAsymTx = 6
Cy3RingAsymRx = 7
Cy3RingSymTxHi = 8
Cy3RingSymRxHi = 9
Cy3RingSymTxLo = 10
Cy3RingSymRxLo = 11

# Data Compression - User space instance #0
Dc0Name = "UserDC0"
Dc0AcceleratorNumber = 0
Dc0BankNumber = 2
Dc0IsPolled = 1
Dc0NumConcurrentRequests = 512
Dc0RingTx = 0
Dc0RingRx = 1

# Data Compression - User space instance #1
Dc1Name = "UserDC1"
Dc1AcceleratorNumber = 1
Dc1BankNumber = 2
Dc1IsPolled = 1
Dc1NumConcurrentRequests = 512
```



```
DclRingTx = 2  
DclRingRx = 3
```



Appendix B Glossary

<i>ADF</i>	Acceleration Driver Framework
<i>AHCI</i>	Advanced Host Controller Interface
<i>AP</i>	Application Processor
<i>ASIC</i>	Application Specific Integrated Circuit
<i>Crystal Beach</i>	Codename for a set of chipset functions that allows discrete PCI Express* (PCIe*) adapters to achieve higher performance.
<i>DID</i>	Device ID
<i>DMA</i>	Direct Memory Access
<i>DTLS</i>	Datagram Transport Layer Security
<i>DRAM</i>	Dynamic Random Access Memory
<i>DRGB</i>	Deterministic Random Bit Generator
<i>DSA</i>	Digital Signature Algorithm
<i>ECC</i>	Elliptic Curve Cryptography
<i>EHCI</i>	Enhanced Host Controller Interface
<i>EVP</i>	Envelope (OpenSSL high-level cryptographic functions)
<i>GbE</i>	Gigabit Ethernet
<i>Gladden</i>	Codename for an Intel® architecture mobile CPU
<i>GPIO</i>	General Purpose Input Output
<i>GPL</i>	General Public License
<i>IBV</i>	Independent BIOS Vendor
<i>LPC</i>	Low Pincount Interface
<i>MGF</i>	Mask Generation Function
<i>MSI</i>	Message Signaled Interrupts



<i>NRBG</i>	Non-deterministic Random Number Generator
<i>PCH</i>	Platform Controller Hub. In this manual, a Intel® Communications Chipset 8900 to 8920 Series device that includes standard interfaces and accelerator and I/O interfaces.
<i>RCiEP</i>	Root Complex Integrated Endpoint
<i>RTOS</i>	Real Time Operating System
<i>SAL</i>	Service Access Layer
<i>SATA</i>	Serial Advanced Technology Attachment
<i>SGL</i>	Scatter Gather List
<i>SIO</i>	Serial I/O
<i>SMBus</i>	System Management Bus
<i>SoC</i>	System-on-a-Chip
<i>SPI</i>	Serial Peripheral Interconnect
<i>SR-IOV</i>	Single Root I/O Virtualization
<i>SSL</i>	Secure Sockets Layer
<i>TLS</i>	Transport Layer Security
<i>TRNG</i>	True Random Number Generator
<i>UART</i>	Universal Asynchronous Receiver/Transmitter
<i>UEFI</i>	Unified Extensible Firmware Interface
<i>UHCI</i>	Universal Host Controller Interface
<i>USB</i>	Universal Serial Bus
<i>VPN</i>	Virtual Private Network
<i>WDT</i>	Watch Dog Timer