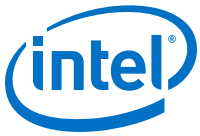
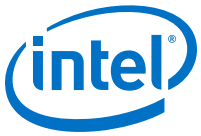
Intel® Communications Chipset 8925 to 8955 Series Software

Programmer's Guide

July 2014

Order No.: 330751-001

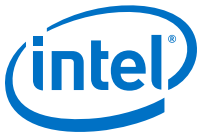




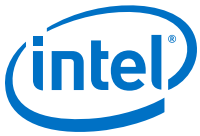
Revision History

Date	Revision	Description
July 2014	001	Updates include: • First "public" version of the document. Based on "Intel Confidential" document number 523126-1.3 with the revision history of that document retained for reference purposes • Added Support for Multiple Acceleration Hardware Generations on page 21 • Added Utility for Loading Configuration Files and Sending Events to the Driver - adf_ctl on page 37 • Updated and added new sections to Heartbeat Feature and Recovery from Hardware Errors on page 49 • Updated Build Flag Summary on page 58 and General Parameters on page 63 • Added Stateless Compression Level Details on page 55 • Added further explanation and images to "decompression service" bullet at end of Intel QuickAssist Technology API Limitations on page 93 • Added PfVfComms Feature Functions on page 123 • Added Reset Device Function on page 125
March 2014	1.3	Updates include: • Added new information to "direct user space access" bullet in Acceleration Drivers Overview on page 32 • Added further detail to note in Hardware Assisted Rings on page 32 • Updated Linux* Software Context for Acceleration Drivers on page 34 • Added Stateless Compression Level Details on page 55 • Added Dynamic Compression for Data Compression Service on page 96, Maximal Expansion with Auto Select Best Feature for Data Compression Service on page 97, and Maximal Expansion and Destination Buffer Size
December 2013	1.2	Updates include: • Added new information to Intel QuickAssist Technology API Limitations on page 93 • Changed document and software title (expanded SKU range to include "8955")
October 2013	1.1	Updates include: • Added NRBG and DRBG information to Random Number Generation Functions on page 112
October 2013	1.0	Corresponds with software release 1.0. Updates include: • Removed two stateful compression/decompression limitations from Intel QuickAssist Technology API Limitations on page 93 • Changed product branding
June 2013	0.5	Corresponds with software release 0.5

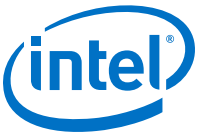
Contents



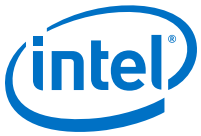
Part 3: Acceleration Drivers.....31



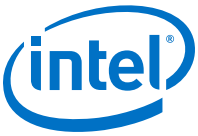
Figures



Tables



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 on page 12).

1.1 Terminology

In this document, for convenience:

- Software package is used as a generic term for the Intel® Communications Chipset 8925 to 8955 Series software package.
- Accelerator is used as a generic term for the Intel® QuickAssist Accelerator integrated in the Intel® Communications Chipset 8925 to 8955 Series PCH.
- Acceleration driver is used as a generic term that allows the Intel® QuickAssist Software Library APIs to access the Intel® QuickAssist Accelerator device(s) integrated in the Intel® Communications Chipset 8925 to 8955 Series PCH.

Refer to Glossary on page 151 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.

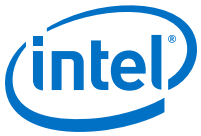
A glossary of the terms and acronyms used in this guide is provided at the end of the document.

1.3 Product Documentation

Documentation supporting the software package includes:

- *Intel® Communications Chipset 8925 to 8955 Series Software Release Notes*
- *Intel® Communications Chipset 8925 to 8955 Series Software for Linux* Getting Started Guide*
- *Intel® Communications Chipset 8925 to 8955 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*

Other related documentation:

- *Intel® Communications Chipset 89xx Series External Design Specification (EDS)*
- *Using Intel® Virtualization Technology (Intel® VT) with Intel® QuickAssist Technology Application Note*
- *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

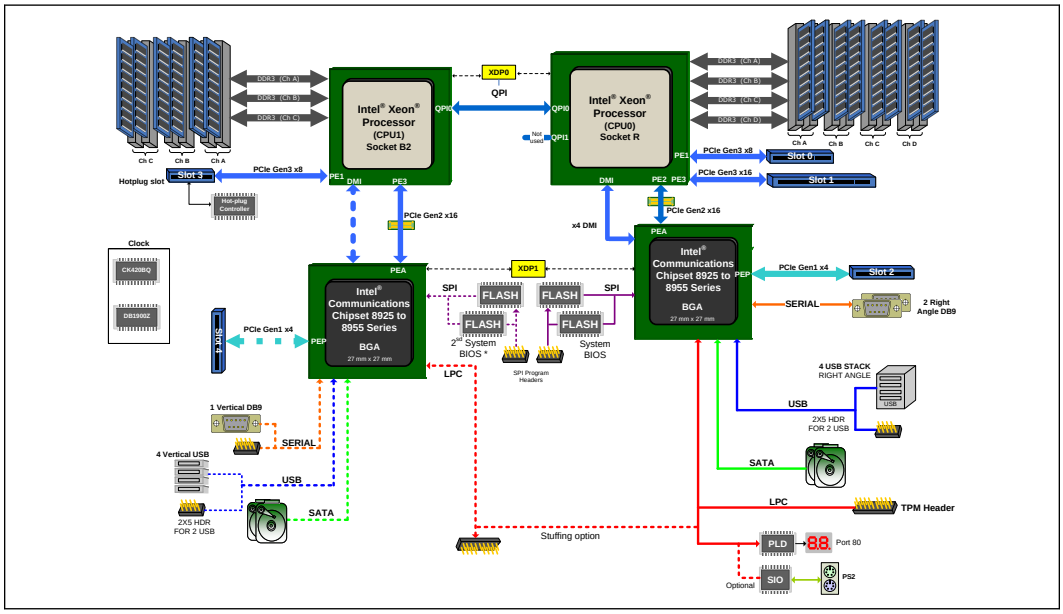
The platform described in this manual is a follow on to previous generation platforms that continue to reduce power, reduce footprint and increase performance for communications infrastructure systems. The platforms deliver leadership solutions with Intel® QuickAssist Technology hardware: the acceleration 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 8925 to 8955 Series chipset. Functionally, Intel® Communications Chipset 8925 to 8955 Series chipset 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).

- Shumway with Intel® Communications Chipset 8925 to 8955 Series (see Figure 1 on page 14) is a next-generation communications platform that features the Intel® Xeon® Processor E5-2658 and E5-2448L with Intel® Communications Chipset 89xx Development Kit.

Figure 1. Shumway with Intel® Communications Chipset 8925 to 8955 Series 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:

```
# lspci -d 8086:0435
```

```
03:00.0 Co-processor: Intel Corporation Device 0435
82:00.0 Co-processor: Intel Corporation Device 0435
```

This displays the PCI configuration space for the 0435 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:

```
# 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 last element of the `0x00040` offset using 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:

```
00101000
```

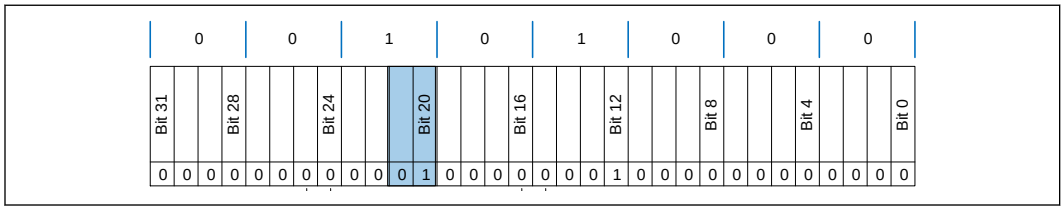
Example

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	21:20 = 00 21:20 = 01 21:20 = 10 21:20 = 11	SKU1 -> DH8925CL SKU2 -> DH8955CL SKU3 -> DH8926CL SKU4 -> DH8950CL

If the 0x00101000 output from the command is analyzed in binary form as shown in the following figure, it can be determined that bits 21:20 are 01, indicating SKU 2.

Figure 2. PCH SKU Identification Example



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 35 04 06 04 10 00 00 00 40 0b 10 00 00 00
```

[0x08] (in bold face above) = 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.

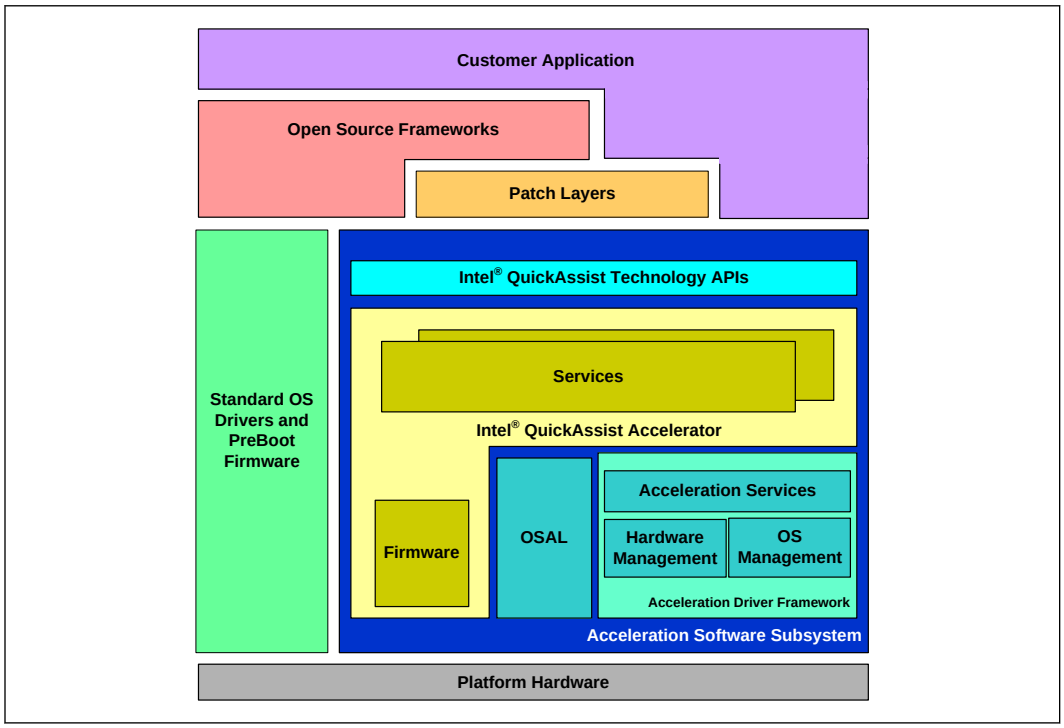
3.0 Software Overview

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 3. Software Architecture Overview



The main software components are:

- **Pre-boot Firmware**

  The (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). These standard OS drivers are described in Part 2: of this manual.

- **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 Part 3: Acceleration Drivers on page 31 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 Part 3: “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 62 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.

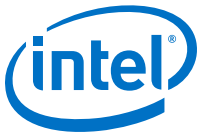
- **Patch Layers**

  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 PCHwith 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

Each logical instance may be configured to operate in one of two modes:

- Interrupt mode
- Polled mode

3.2.1.1 Interrupt Mode

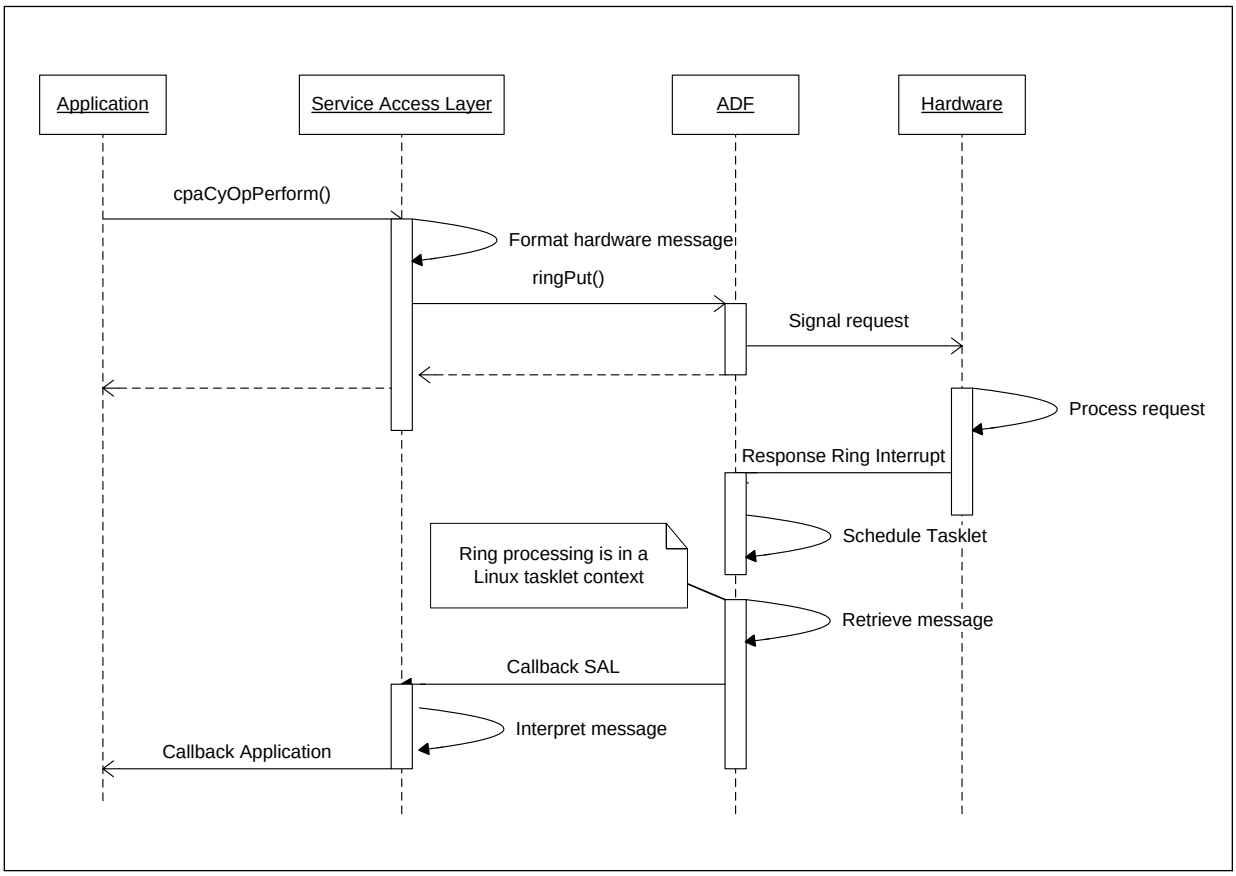
When configured in interrupt mode, the Accelerator Driver Framework (ADF) registers an interrupt handler for response ring processing.

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 half[1] 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 137 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 4. 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 or *DcXIsPolled*attribute of a logical instance in the configuration file to 1. See Cryptographic Logical Instance Parameters on page 68 and Data Compression Logical Instance Parameters on page 69 for more information. When configured to 1, the ADF does not service interrupts for that logical instance.

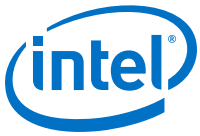
The ADF provides a set of APIs to allow the client to poll a single bank or all banks on a given accelerator:

- 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* operating system.

3.4 OpenSSL* Library Inclusion and Usage

The Intel® Communications Chipset 8925 to 8955 Series Software 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 Intel® Communications Chipset 8925 to 8955 Series Software 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 37 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:

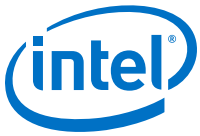
- If one or more Intel® Communications Chipset 8900 to 8920 Series devices are visible on the PCIe bus and no Intel® Communications Chipset 8925 to 8955 Series device is present, the installer.sh will build with support for Intel® Communications Chipset 8900 to 8920 Series devices only.
- If one or more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



Part 2: Core and Chipset Drivers

4.0 Embedded Drivers

In general, the software package can be described as containing two kinds of drivers:

- Embedded Drivers - These drivers are enumerated in this chapter.
- Acceleration Drivers - These drivers are described in Acceleration Drivers Overview on page 32.

4.1 Overview

The platform supports the following embedded drivers:

- USB
- SATA (supports two ports)
- LPC (includes WDT and Serial I/O)
- SPI
- GPIO
- Crystal Beach DMA (server platform only)
- Non-Transparent Bridge (server platform only)

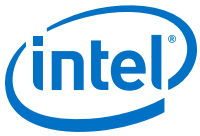
When more than one PCH device is present on a platform, only one of the PCH devices has the standard PC drivers enabled; the others make only the PCIe* end-point visible.

4.2 USB Drivers

The PCH provides one EHCI USB2 Host Controller with six ports. The Enhanced Host Controller Interface (EHCI) provides a standard register interface to USB 2.0. There is also the ability to access these same six ports via the Universal Host Controller Interface (UHCI), the previous generation register interface, which only supports USB 1.1. The following features are provided:

- USB Rate Matching Hub
- Two debug ports
- Supports wake up from S1-S5
- Legacy keyboard/mouse software with USB keyboard/mouse
- Per port USB disable
- VCp for isochronous traffic (VC0 for asynchronous)
- Capability to use reduced Frame List Sizes
- Support for hot plug and surprise removal

The following limitations apply:



- USB-Redirect, which provides the ability for a remote management agent to gain access through the NIC and act as if it were a local USB device (typically a keyboard or mouse), is not supported.
- *USB on the Go* is not supported in the PCH.

4.3 SATA Drivers

The PCH provides up to two SATA Controllers, supporting two SATA Ports. Advanced Host Controller Interface (AHCI), the SATA standard register interface, is supported (on one function, as described below).

The features are as follows:

- Integrated DMA operations on two ports
- SATA Gen2 support, 300 MB/s on each port
- Per port activity LEDs
- Multiple MSI message vectors
- Dynamic AFE Squelch
- Legacy IDE software interface supported as configuration option (in BIOS)

Two modes are supported in the SATA Controller:

- AHCI
- Legacy IDE

When in AHCI mode, the SATA Controller only exposes one PCI function, Device 31 Function 2 (D31F2). When in Legacy IDE mode, an additional function is exposed, Device 31 Function 5 (D31F5).

This is controlled through the register offset 90h MAP, Port Mapping Register. Bit 5 is the SATA Port to controller Config register (SC).

- When this bit is '0' (Legacy Mode):
  - Up to four SATA ports are in the D31F2 controller with port[3:0]. In the PCH, none of these ports are enabled.
  - Up to two SATA ports are available in the D31F5 controller with port[5:4] (according to SATA pin list). These are the two implemented SATA ports.
- When this bit is '1' (AHCI Mode):
  - Up to six SATA ports are in the D31F2 controller with port [5:0]. Only Ports 4 and 5 are enabled.

No SATA port is available in the D31F5 controller. For operation in IDE mode, this bit should be '0'. Legacy Mode offers less performance than AHCI mode and therefore should only be used in OSs where AHCI is not available.

In AHCI mode, it is the AHCI Port Disable bit that allows a driver to know if a given SATA Port exists (this is in the Port Mapping Register). Therefore, in the Intel® Communications Chipset 8925 to 8955 Series PCH, Ports 0 through 3 are disabled.

4.4 LPC Device

The PCH provides the Low-Pincount (LPC) interface. This interface:

- Allows connection of devices such as Super I/O, micro controllers, customer ASICs
- Supports two master/DMA devices
- Uses a memory size up to 8 MB

In addition, the WDT and Serial I/O are integrated into the LPC. Note that there is no separate LPC driver as such, instead, there are the drivers for the devices on the LPC bus, specifically separate drivers for WDT and Serial I/O.

4.4.1 Watch Dog Timer Drivers

The PCH includes Serial I/O and Watch Dog Timer (WDT) as part of the LPC. The WDT features are as follows:

- 33 MHz Clock (30 ns clock ticks)
- Multiple Modes (WDT and Free-Running)
- Timer can be disabled (default state) or Locked
- WDT Automatic Reload of Preload value when WDT Reload Sequence is performed

Note: The WDT driver is not part of any standard Linux* distribution and is provided as sample code only.

Note: In addition to the WDT described above, there are two other watch dog entities available in the system:

- TCO Watch dog (Total Cost Ownership/System Management Watch dog); a kernel patch has been submitted for this driver against Linux kernel version 2.6.xx
- Per-Thread watch dog - (device ID 0x2360)

4.4.2 Serial I/O Drivers

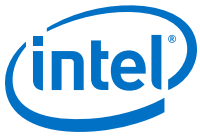
The serial I/O has the following features:

- Two Full Function 16550 Compatible Serial Ports
- Configurable I/O addresses and interrupts
- 16-Byte FIFOs
- Supports up to 115 Kbps
- Programmable Baud Rate Generator
- Modem Control Circuitry
- 14.7456 MHz, 33 MHz, and 48 MHz supported for UART baud clock input

4.5 SPI Drivers

The PCH supports a single SPI interface. The SPI is used to connect the Flash device used to boot the system. Its features include:

- Supports up to two 16 MB devices (two chip selects)
- Supports the SPI Fast Read instruction
- Hardware decompression for Acceleration Engine Sx Operation



Note: The SPI drivers are not part of any standard Linux* distribution and are provided as sample code only.

4.6 GPIO Drivers

The PCH supports GPIO pins some of which are available for customer use. See the *External Design Specification* for more information.

A GPIO driver is not provided. Instead, illustrative code is provided that shows how the GPIOs can be used. See the *General Purpose I/O (GPIO) Use in Software Application Note* for more information.

4.7 Crystal Beach DMA Application

Note: The Crystal Beach (CB) DMA application is supported on server platforms only.

Crystal Beach (CB) technology provides a set of chipset functions that allow discrete PCI Express* (PCIe*) adapters to achieve higher performance while decreasing adapter cost. The main features of CB are as follows:

- Supports write operations from memory to I/O, but not from I/O to memory
- Instantiated as a root complex integrated PCIe end point device
- Chipset DMA that is controllable by software executing on the processor
- PCI Express enhancements such as relaxed ordering
- A standardized software interface for controlling and accessing DMA features
- One MSI or MSI-X vector supported per CB channel/function
- SR-IOV support is not provided in the hardware
- Support for Asynch_tx on the CB driver

There are eight software visible CB DMA engines, visible as PCI functions. Each engine has one channel. Each can be independently operated, and in a virtualized system each can be independently assigned to a VM. In the PCH, all eight channels are DMA engines.

For Linux*, Crystal Beach uses *async_tx*. Refer to the Asynchronous Transfers/ Transforms API document for a description. Other operating systems are not supported.

Note: The CB DMA application is not part of any standard OS distribution and is provided as sample code only.

For more information on the Crystal Beach (CB) DMA feature, see the *Intel® Xeon® Processor (storage) - External Design Specification (EDS) Addendum - Rev. 1.1* (Reference: 503997).

4.8 Non-Transparent Bridge (NTB) Driver

Note: The Non-Transparent Bridge (NTB) driver is only supported on Shumway.

On server platforms, one of the root ports may be converted to a Non-Transparent Bridge (NTB) device interface. While a transparent bridge simply forwards requests and responses from one side to the other using the PCIe* header for routing, an NTB is used to isolate one root complex from another and to selectively allow specific memory range forwarding.

In a typical system, when an NTB is enabled, it exposes primary and secondary sides to the host and remote systems respectively. The NTB is seen as a Root Complex Integrated Endpoint (RCiEP) from the primary side. As such, the NTB device behaves mainly as a PCIe endpoint device with a couple of different rules as follows:

- It does not support OS power management that is separate from the chipset
- It cannot support I/O (as opposed to MMIO) requests

The BIOS will configure the PCIe port as one of the following possible configurations:

- A PCIe root port
- An NTB that is connected to a second NTB on another system, called back-to-back (B2B)

The software package includes a set of device drivers provided as sample code for use by the client software to support each of the NTB configurations.

The NTB device exposes a Type-0 PCIe configuration space on each side. The upstream side nearest the CPU is visible as a Type-0 Root Complex Integrated Endpoint (RCiEP) and the downstream secondary side exposes itself to another system as a PCIe Endpoint (EP).

Note: The NTB driver in the software package is a modified version of the NTB driver that has been upstreamed to the later Linux kernel 3.9. Fedora 16 uses an earlier kernel version.

See the Intel® Xeon® Processor C5500/C3500 Series Non-Transparent Bridge Programmer's Guide for more information.

4.9 Intel Technology Support

The platforms described in this manual support the following Intel technologies:

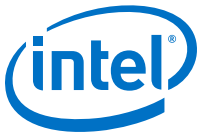
- Intel® Virtualization Technology (Intel® VT)
- Intel® 64 architecture
- Intel® Simultaneous Multi-Threading (Intel® SMT)

See the following topics for short descriptions and pointers to more detailed information.

4.9.1 Intel® Virtualization Technology (Intel® VT)

Hardware-assisted Intel® Virtualization Technology (Intel® VT) provides greater flexibility and maximum system utilization by consolidating multiple environments into a single server, workstation, or PC. With fewer systems required for the same tasks, Intel® VT delivers:

- Simplified resource management, increasing IT efficiency.



- Greater systems reliability and availability, reducing corporate risk and real-time losses from downtime.
- Lower hardware acquisition costs with increased utilization of the machines you already have.

The platforms described in this manual support the following:

- Intel® Virtualization Technology (Intel® VT-x); (see http://www.intel.com/technology/virtualization/technology.htm)
- Intel® Virtualization Technology for Directed I/O (Intel® VT-d); (see http://www.intel.com/technology/itj/2006/v10i3/2-io/1-abstract.htm)
- Intel® Virtualization Technology for Connectivity (Intel® VT-c); (see http://www.intel.com/network/connectivity/solutions/vmdc.htm)
  - Virtual Machine Device Queues (VMDq); (see http://www.intel.com/network/connectivity/vtc_vmdq.htm)

Intel® VT also complements the Single Root I/O Virtualization and Sharing (SR-IOV) specification created by the Peripheral Component Interconnect Special Interest Group* (PCI-SIG*).

The acceleration driver supports simultaneous access of the acceleration hardware from a Virtual Machine (VM) through a Virtual Function (VF) and a Virtual Machine Manager (VMM) through a Physical Function (PF).

For specific detail, see the *Using Intel® Virtualization Technology (Intel® VT) with Intel® QuickAssist Technology Application Note*.

4.9.2 Intel® Simultaneous Multi-Threading (Intel® SMT)

Intel® Simultaneous Multi-Threading (Intel® SMT) technology is an architectural feature of a processor that allows multiple threads to issue instructions on each cycle. In other words, SMT allows the functional units that make up the processor to work on behalf of more than one thread at the same time.

4.9.3 Intel® 64

Intel® 64, formerly known as Intel® Extended Memory 64 Technology (EM64T), allows server, workstation, and desktop platforms to access larger amounts of memory. This enhancement allows a processor to run newly written 64-bit code and access larger amounts of memory than 32-bit code. Intel 64 is often referred to as "64-bit extensions" to the Intel architecture 32-bit (IA-32).

See http://www.intel.com/technology/intel64/index.htm for more information.

4.10 Other Supported Technologies and Standards

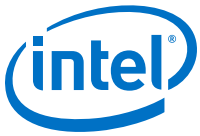
The platforms described in this manual also support:

- Intel® AES New Instructions (Intel® AES-NI) - See http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni/ for details.

  Note: AES denotes Advanced Encryption Standard.
- Intel® Advanced Vector Extensions (Intel® AVX) - See http://software.intel.com/en-us/avx/ for more information.

- Asynchronous DRAM Refresh (ADR) - See the white paper at http://pcache-www.intel.com/cd/00/00/45/60/456090_456090.pdf for more information.

  Note: ADR is supported on the server platforms only.



Part 3: Acceleration Drivers

5.0 Acceleration Drivers Overview

In general, Intel® Communications Chipset 8925 to 8955 Series Software can be described as containing two kinds of drivers:

- Embedded Drivers - These drivers are described in Embedded Drivers on page 24.
- 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.

5.1 Hardware Assisted Rings

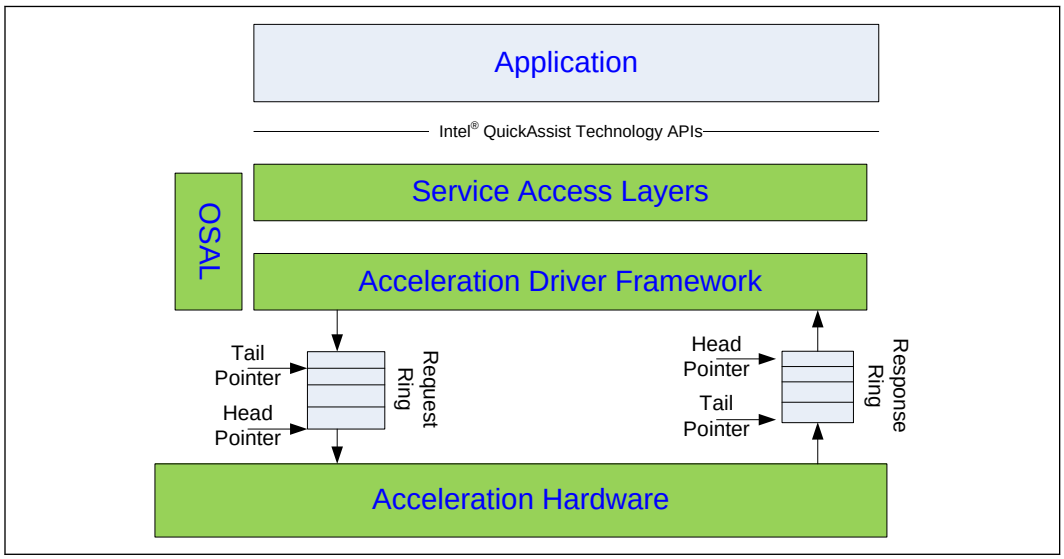
Hardware assisted rings are used as the communication mechanism to transfer requests between the CPU and the accelerator(s) on the chipset device and vice-versa. The hardware supports 512 rings, each with head and tail Configuration Status Register (CSR) pointers that are mapped to PCIe* memory on the 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 rings have a default size of 512 entries each (request and response). 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 5. Intel® QuickAssist Accelerator Ring Access



Rings are grouped into *ring banks* with each ring bank containing 16 rings.

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

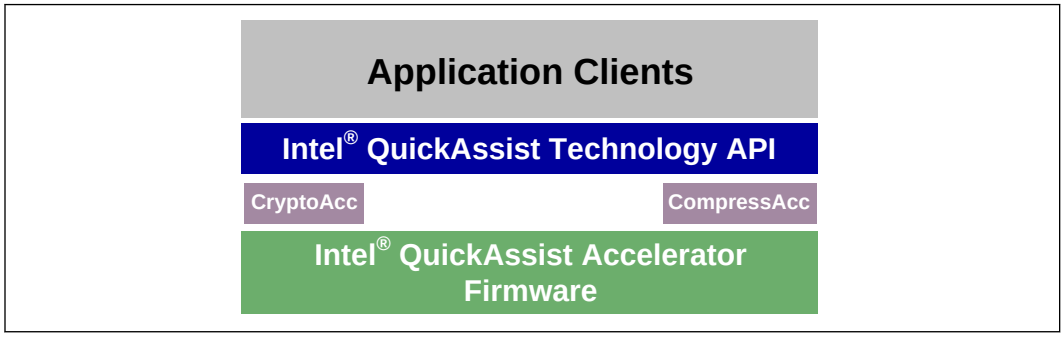
MSI-X interrupts are supported by the Intel® QuickAssist Accelerator, and if the OS supports MSI-X interrupts, the response may be directed to any core on system. This allows an even distribution of response processing among the cores on the system. The configuration of bank interrupts and core affinity is detailed in Affinity Parameters on page 138.

All rings on the device are shared by the Intel QuickAssist Accelerators on the device. The hardware load balances requests from these rings across the Intel QuickAssist Accelerators.

5.2 Basic Software Context for Acceleration Drivers

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

Figure 6. Basic Software Context



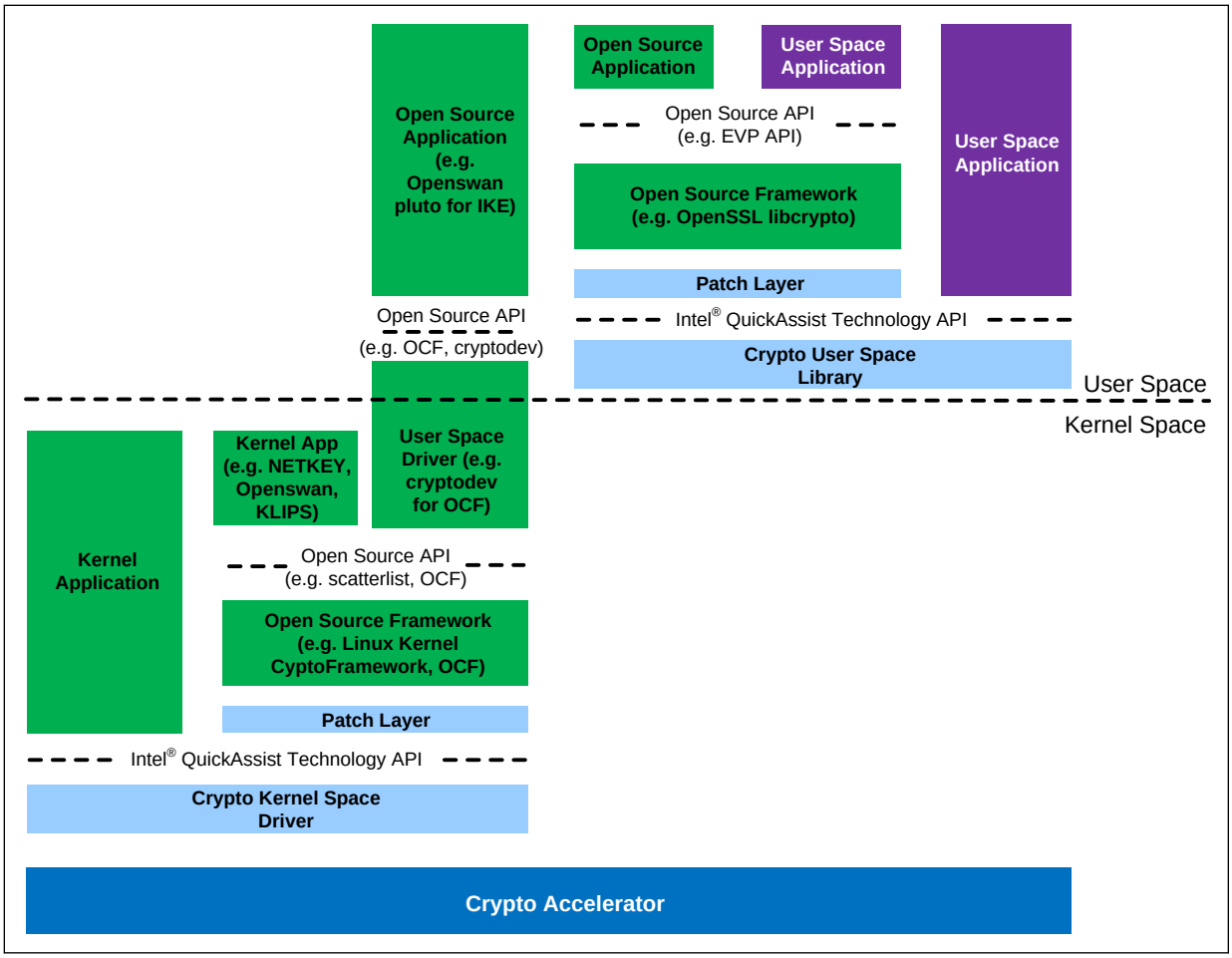
The key elements of this model are as follows:

- 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.

5.3 Linux* Software Context for Acceleration Drivers

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

Figure 7. 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.

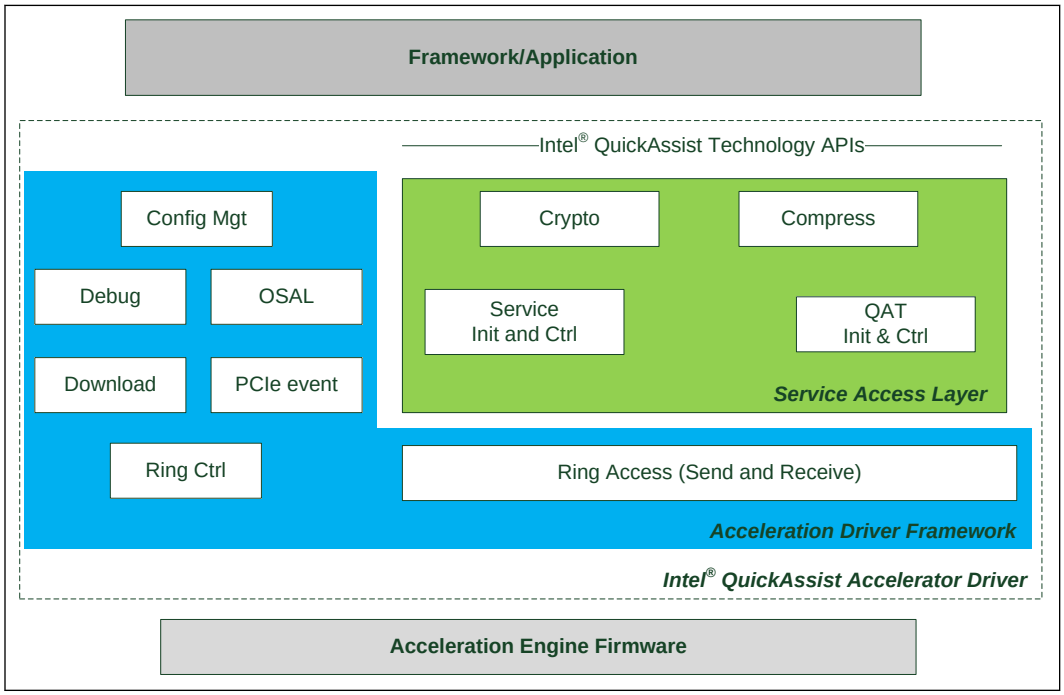
Note: The applications identified in the figure above are examples only and do not serve as an statement of intent for enabling.

Note: Software packages for patches, such as OpenSSL, Linux Kernel Crypto Framework, and NetKey and zlib are distributed separately. See Product Documentation on page 12. You will need an Intel Business Link (IBL) account and a subscription to the Electronic Design Kit (EDK).

5.4 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 8. Acceleration Driver Framework



5.4.1 Framework Overview

An acceleration driver contains a number of logical units that are primarily exposed via the Intel® QuickAssist Technology APIs. Figure 8 on page 35 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.

5.4.2 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 8 on page 35, 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:

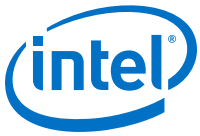
- Register for notification of, query, observe and handle initialization/discovery/error events from the ADF framework. The layer initializes and stops services based on the state of the accelerator as indicated by ADF.
- Initialize the service layers based on the settings in a configuration file.
- 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.

5.4.3 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.

5.4.4 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 62. The older legacy configuration file format (which is still supported) is described in Acceleration Driver Configuration File - Earlier File Format on page 136.

5.4.5 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 DH895xccs, two DH89xxccs 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 icp_sal_poll Bank, etc.**	**Passed by mux to driver in call to icp_sal_poll Bank, etc**
	accelId	*hw_data.dev_class.name*	*hw_data.InstanceId*		*accelId on API*	*accel_dev.accelId in driver*
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 8925 to 8955 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`

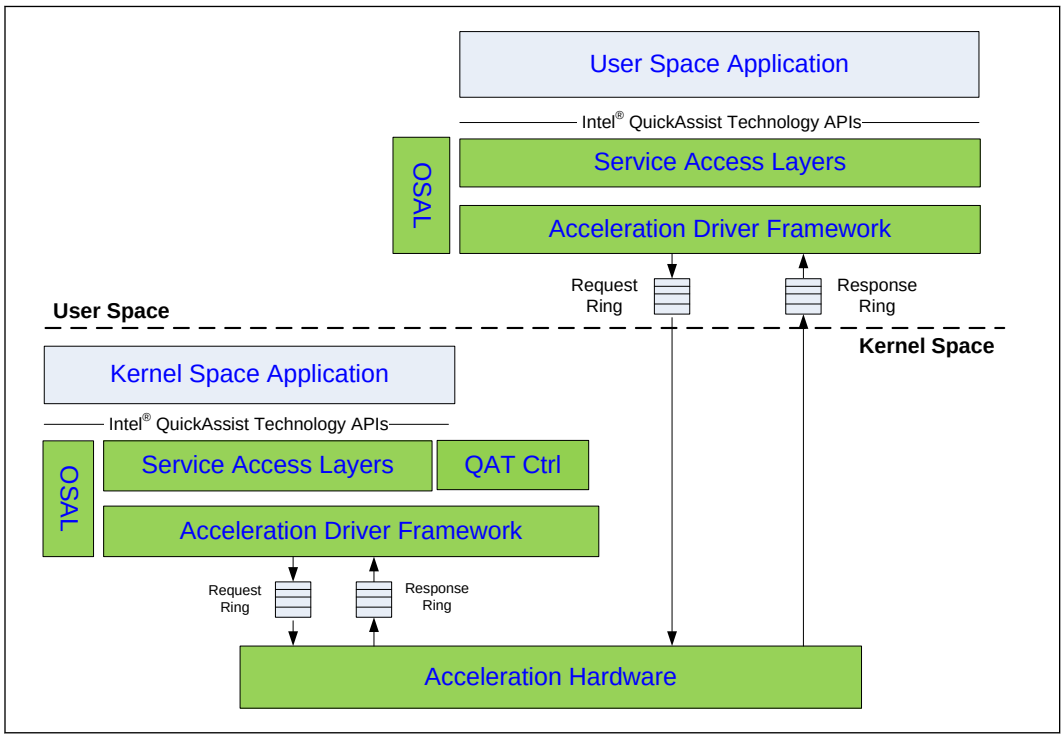
Case where DH895xcc and DH89xxcc devices are on the platform

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/cpa_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`

5.5 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 9. 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.

5.5.1 Communication Between User Space and Kernel Space Drivers

The QAT kernel space driver creates several Linux* device drivers as a means of interacting with the QAT user-space driver that is linked in to client user-space processes. The paths to the Linux device drivers vary depending on which QAT driver is loaded as indicated in the following table.

QAT1.5 driver	QAT1.6 driver, if not built for mux. (and so QAT1.5 can/will not be loaded on this platform)	QAT1.6 driver, if built for mux. (and so QAT1.5 may be loaded on this platform)
/dev/icp_adf_ctl	/dev/icp_adf_ctl	/dev/icp_mux/icp_adf_ctl
/dev/icp_devX_csr	/dev/icp_devX_csr	/dev/icp_mux/icp_devX_csr
/dev/icp_devX_ring	/dev/icp_devX_ring	/dev/icp_mux/icp_devX_ring
/dev/icp_dev_processes	/dev/icp_dev_processes	/dev/icp_mux/icp_dev_processes
/dev/icp_dev_mem	/dev/icp_dev_mem	/dev/icp_mux/icp_dev_mem
	/dev/icp_dev_pfvfcomms	/dev/icp_mux/icp_dev_pfvfcomms

These drivers are typically used at driver and device initialization, rather than on the data path, with the exception of `icp_dev_ring` which is used for user-space interrupt processing. For maximum performance on the data path, the user-space driver accesses memory mapped into user space or accesses the device directly.

5.5.2 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

5.5.2.1 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.

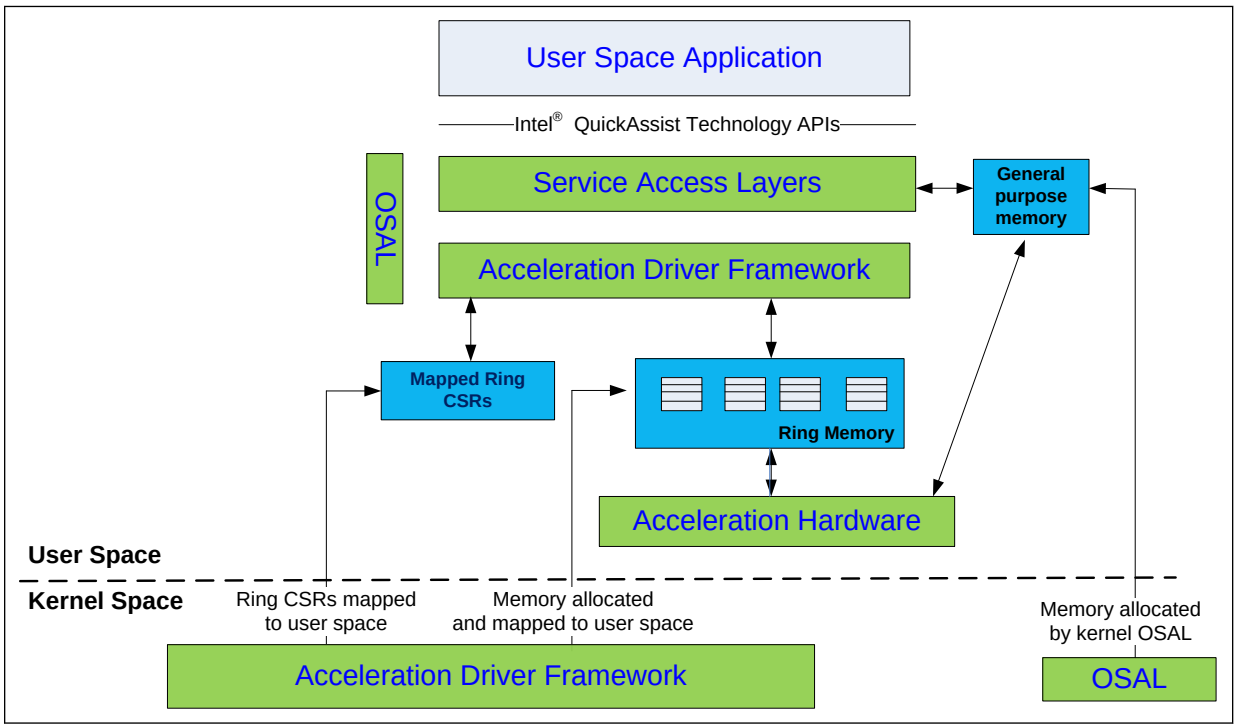
In user space, it is a little more complex. The OSAL implementation of `osalMemAllocContiguousNUMA` needs to return memory that is resident and contiguous. To do this, the OSAL in kernel space creates a device, called `icp_dev_mem` that may be called through an `IOCTL` function by the OSAL in user space 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_ROOT/quickassist/utilities/osal/include/Osal.h` for details) with your specific implementation.

Figure 10. User Space Memory Allocation at Initialization



5.5.2.2 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	`cpaCySetAddressTranslation`	See the *Intel® QuickAssist Technology Cryptographic API Reference Manual* for details.
Data Compression service	`cpaDcSetAddressTranslation`	See the *Intel® QuickAssist Technology Data Compression API Reference Manual* 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.

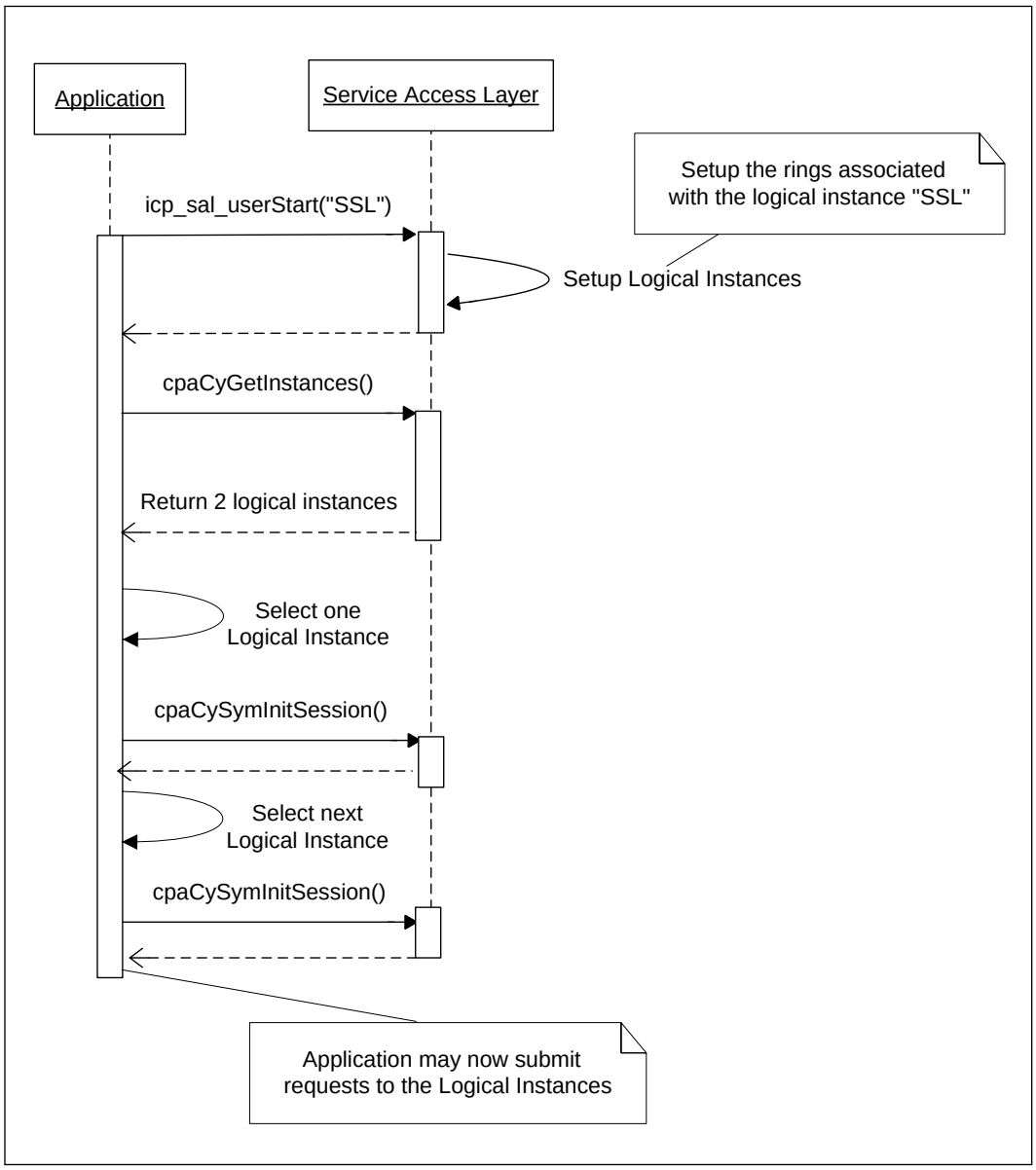
5.5.3 User Space Additional Functions

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 43 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 119 function (or icp_sal_userStart on page 118 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 120.

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 11. User Space Process with Two Logical Instances



5.5.4 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 11 on page 43, 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
# List of core affinities
Cy0CoreAffinity = 0,1

# Crypto - User instance #1
Cy1Name = "SSL1"
Cy1IsPolled = 1
# List of core affinities
Cy1CoreAffinity = 2,3
```

In this example, the user process SSL configures two logical instances (called ”SSL0” and ”SSL1”).

5.5.5 User Space Response Processing

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

- Interrupt mode
- Polled mode

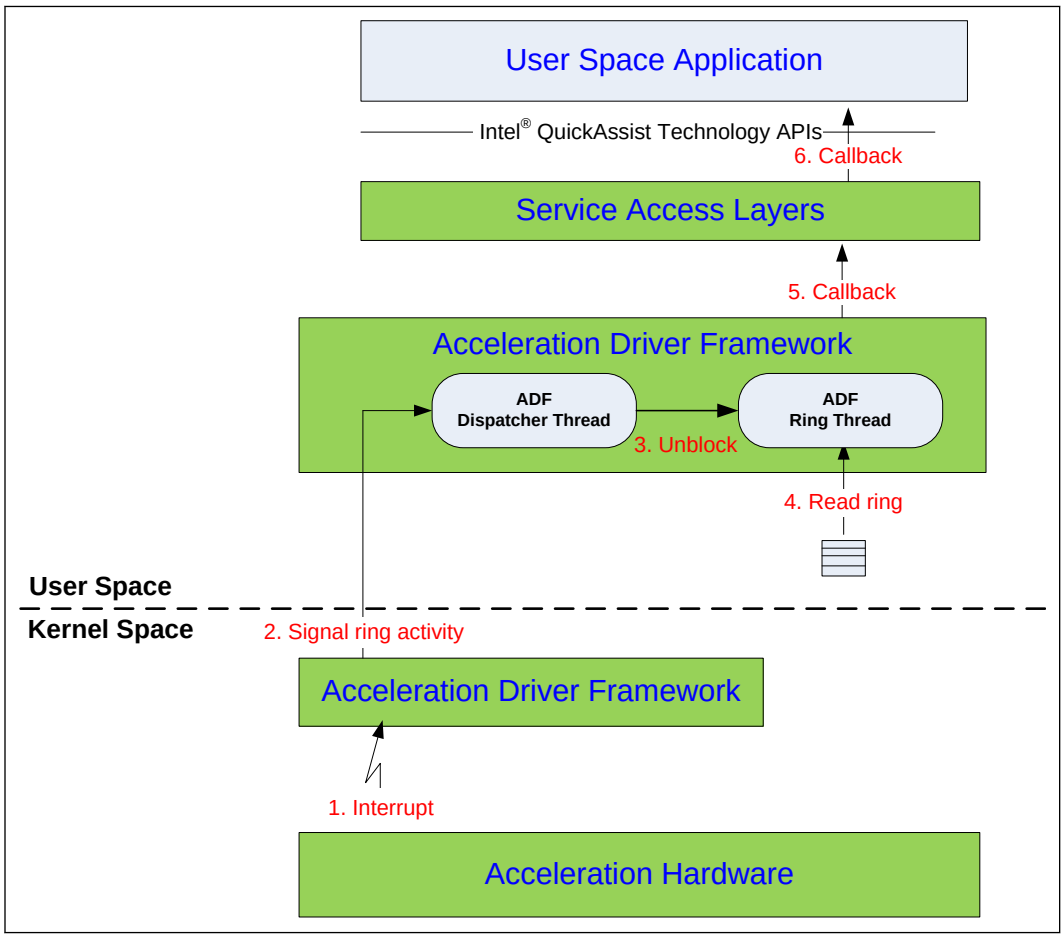
5.5.5.1 User Space Interrupt Mode

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 12. User Space Response Processing for Interrupt Mode



5.5.5.2 User Space Polled Mode

The sequence for user space polling does not differ from that described in Polled Mode on page 20.

5.6 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.

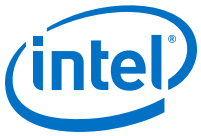
The *shutdown* qualifier enables the user to bring down all devices and unload driver modules from the kernel. This contrasts with the *stop* qualifier which brings down one or more devices, but does not unload kernel modules, so other devices can still run.

5.7 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 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 65 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_dh895xcc_dev0/cy/
total 0
-r--------. 1 root root 0 Jun 21 14:18 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
```

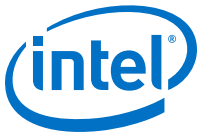
The statistics can then be queried simply by running `cat` on the corresponding file in the `/proc` file system. For example:

```
# cat /proc/icp_dh895xcc_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 |
+-------------------------------------------
```



5.8 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_AUTO_DEVICE_RESET 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.

When an heartbeat query is triggered by the user, the driver sends a 'SYNC' message and after a pre-defined time sends a 'GET' message which returns a bit that indicates if the device is blocked/not blocked. The firmware, if healthy, responds to the heartbeat query reporting its state (blocked or not blocked). If the firmware is not responsive a timeout occurs. The heartbeat query at low level is converted in two heartbeat init/admin messages (SYNC and GET).

The Heartbeat feature can be used to poll the firmware to check for liveness using any of the following methods:

- Periodically call heartbeat APIs, that is, `icp_adf_check_device()` or `icp_adf_check_all_devices()`.
- Watch on cat /proc/icp../qat or /proc/icp../heartbeat

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` or the `icp_reset_device()` API.

5.8.1 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. The user should apply the desired method to each device of interest.

- Manually from command line using the command:

```
# cat /proc/icp_dh895xcc_dev0/heartbeat
```

- From a watch process running in background:

```
# watch -n0.1 cat /proc/icp_dh895xcc_dev0/heartbeat > /dev/null
```

- From simple script running in the background:

```
#!/bin/bash
while :
do
  cat  /proc/icp_dh895xcc_dev0/heartbeat > /dev/null
  sleep 1
done
```

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

```
# cat /proc/icp_dh895xcc_dev1/heartbeat
```

If the device is functioning properly, the following message is displayed:

```
Device up and running!
```

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
```

5.8.2 User Application Heartbeat APIs (not Enabled by Default)

Anytime after the initialization process, that is, after a call to either `icp_sal_userStart()` or `icp_sal_userStartMultiProcess()`, the customer application may periodically call either the `icp_sal_check_device()` or the `icp_sal_check_all_devices()` function to perform the check of the firmware/hardware on a given Acceleration device or on all Acceleration devices, respectively.

These functions have the following signatures:

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

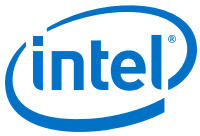
```
CpaStatus icp_sal_check_all_devices(void);
```

See icp_sal_check_device on page 121 and icp_sal_check_all_devices on page 122 for details on the functions and parameters.

5.8.3 Handling Heartbeat Failures

The driver must be compiled with ICP_AUTO_DEVICE_RESET 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 49.
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.

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

5.8.3.1 AER and Uncorrectable Errors

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

- **Uncorrectible 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.

The reset is only done if the driver is built with the ICP_AUTO_DEVICE_RESET compiler flag.

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

5.8.4 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 49 or User Application Heartbeat APIs (not Enabled by Default) on page 50).
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 Heartbeat feature 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.

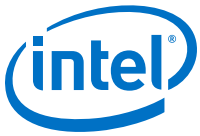
Note: If the heartbeat feature in the acceleration software is not enabled, the PF driver will not notify the VF driver that the firmware is unresponsive.

Device errors requiring a device reset (Secondary Bus Reset or SBR) can be detected by the Host using the Heartbeat, Uncorrectible Error and AER features. Typically the Host application running on the PF will want to control the timing of any SBR. Even though an SBR may be necessary to recover from errors, the Host may delay this so it can communicate with VMs, allowing them to gracefully manage the errors until the Host resets the device. Resetting one device can have knock-on effect on the VM forcing it to restart and affecting all other functionality provided by the VM, e.g., if the SBR is delayed in a system with multiple acceleration devices the VMs may divert traffic away from the affected device to another device and so continue to provide service with reduced capacity. Later at a quiet time, e.g., in the middle of the night, the Host can reset the device and the affected VMs can be restarted

To allow the Host to control device reset timing the driver must be built without the ICP_AUTO_DEVICE_RESET flag.

A typical heartbeat error use-case in a virtualized system:

1. The PF driver is loaded, initialized and started in the Host.
2. The VF driver is loaded, initialized and started in the Guest OS on the VM(s).
3. The Host user-space application detects that the firmware is unresponsive using the heartbeat feature (see Heartbeat Feature and Recovery from Hardware Errors on page 49) in the PF driver.



4. The Host application communicates with the Guest application(s) on the VM(s) using the Intel® QuickAssist Accelerator driver's PfVfComms feature (see PfVfComms Feature Functions on page 123
5. The Guest and Host applications takes whatever steps are necessary to stop using the errored device.

   Sometime later...
6. The Host application calls a device reset using the `icp_reset_device()` API or `adf_ctl` utility.
7. The PF kernel-space driver sends the Restarting event to any user-space processes on the Host.
8. The PF driver sends the Restarting event message to any VFs which are up, via the PfVfComms mechanism. Note there may not be any VFs up at this stage, as Guest applications may have used the previous communication to bring the device down.
9. On any VFs which are still up the VF kernel-space driver sends the Restarting event to any user-space processes.
   - The user-space processes pass it on to the Guest application's registered callback.
   - The Guest application may gracefully shutdown.
   - The Guest OS may gracefully shutdown.

   Note: The PF does not wait until VFs have completed any actions, once the Restarting message has been received on all VFs it goes on to next step.
10. The PF driver triggers the device reset (save state, initiate SBR, restore state).
11. The Host application restarts the Guest OS and loads the VF driver and application in the Guest.

If the PF driver is built with the ICP_AUTO_DEVICE_RESET flag, steps 4, 5 and 6 are skipped and there is no delay between error detection and device reset.

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.

5.9 Driver Threading Model

By default, when an application uses the acceleration driver (`libicp_qa_al_s.so`) 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.

5.9.1 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.

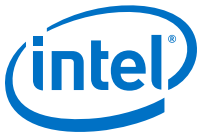
It is the user's responsibility to use these functions to monitor the state of devices and receive device-related events.

5.10 Stateful 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.

5.11 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: As highlighted in the following table, compression levels 3-9 are the same for the 32- and 8-Kbyte History Windows.

Note: The State registers are also saved.

History Windows	Compression Level	Search Depth	Context Size
32 kB	1	1	0
32 kB	2	1	48 kB
32 kB	3	4	0
32 kB	4	8	0
32 kB	5	8	32 kB
32 kB	6	8	40 kB
32 kB	7	16	0
32 kB	8	16	32 kB
32 kB	9	16	40 kB
8 kB	1	4	0
8 kB	2	4	32 kB
8 kB	3	4	40 kB
8 kB	4	8	0
8 kB	5	8	32 kB
8 kB	6	8	40 kB
8 kB	7	16	0
8 kB	8	16	32 kB
8 kB	9	16	40 kB

5.12 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: As highlighted in the following table, compression levels 3-9 are the same for the 32- and 8-Kbyte History Windows.

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

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

continued...

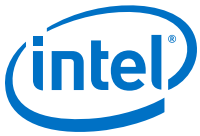
History Windows	Compression Level	Search Depth	Context Size (Kbyte)
32 kB	5	8	0
32 kB	6	8	0
32 kB	7	16	0
32 kB	8	16	0
32 kB	9	16	0
8 kB	1	4	0
8 kB	2	4	0
8 kB	3	4	0
8 kB	4	8	0
8 kB	5	8	0
8 kB	6	8	0
8 kB	7	16	0
8 kB	8	16	0
8 kB	9	16	0

5.13 Acceleration Driver Error Scenarios

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

5.13.1 User Space Process Crash

Error Scenario	A user space process crashes without cleanly stopping the user space acceleration driver in the process.
Background	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 `ioctl` calls on `icp_dev<N>_ring`. The kernel acceleration driver maintains a list of rings per pid, per device. 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 `ioctl` calls on `icp_dev_mem`. The kernel driver keeps track of all memory allocated per pid. These files are opened at initialization when an application calls `icp_sal_userStart*()` and are closed when an application calls `icp_sal_userStop()` or closed by the operating system when the application is killed/crashed.
Sequence of Events	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 `/dev/icp_dev_*` file. 3. The kernel acceleration driver frees any allocated resources (rings/memory) associated with the crashed process. 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



5.13.2 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 an only if the Heartbeat feature is enabled using the ICP_AUTO_DEVICE_RESET compile-time option.
Sequence of Events	Refer to Handling Heartbeat Failures on page 50.

5.13.3 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	1. Acceleration hardware detects an un-correctable error. It notifies the kernel acceleration driver via an error interrupt. 2. If, and only if the automatic device reset feature is enabled by the ICP_AUTO_DEVICE_RESET compile-time option, the kernel acceleration driver resets the acceleration device upon receipt of the interrupt.
Side Effects	Same as Hardware Hang Detected by Heartbeat on page 57.

5.13.4 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 56. There is no involvement from the Intel® QuickAssist Technology Physical Function (PF) driver in this scenario.
Side Effects	None

5.13.5 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 informs the PF driver that it's shutting down. The host OS/VMM then un-assigns 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	1. The VM crashes. 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.

5.13.6 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	See Handling Device Failures in a Virtualized Environment on page 51
Side Effects	All VMs that are assigned VFs from the same silicon device are affected.

5.13.7 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	1. The un-correctable error is reported to the Physical Function (PF) acceleration driver running in the host OS. See Handling Device Failures in a Virtualized Environment on page 51
Side Effects	All VMs that are assigned VFs from the same silicon device are affected.

5.14 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 2. Required Build Flags

Symbol	Description	Default	Reference
ICP_BUILDSYSTEM_PATH	Set to the build system folder located under the quickassist folder (/QAT/quickassist/build_system)	User defined	
ICP_BUILD_OUTPUT	Set to directory that executable/ libraries are placed in (/QAT/build)	User defined	
ICP_ENV_DIR	Set to the directory that contains the environmental build files (/QAT/quickassist/build_system/build_files/env_files)	User defined	
ICP_ROOT	Set to the directory where acceleration software is extracted (/QAT)	User defined	
ICP_TOOLS_TARGET	Set to accelcomp for Intel® Communications Chipset 8925 to 8955 Series Software platforms	User defined	

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

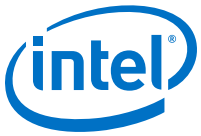


Table 3. Optional Build Flags

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 cpaCyDrbgSessionInit and the DRBG HT functions to poll for responses internally rather than depending on an external polling thread.	Not defined	DRBG Health Test and cpaCyDrbgSessionInit Implementation Detail on page 117
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 icp_sal_userStart or icp_sal_userStartMultiProcess.	Not defined	Thread-less Mode on page 54
ICP_AUTO_DEVICE_RESET	When undefined, the driver will automatically reset the device on detection of any of the following errors: • Heartbeat fail • Uncorrectable error interrupt • Advanced Error Report detected by kernel When defined, the device will not be reset on error detection. The device must be manually reset instead. It is recommended that this be defined for non-virtualized systems and not defined for virtualized systems.	Not defined	
ICP_NONBLOCKING_PARTIALS_PERFORM	When defined, results in partial operations not being blocked.	Not defined	Defined when compiling the driver using the installer.sh installation script.
ICP_SRIOV	Indicates whether SRIOV should be enabled in the driver.	Not defined	
ICP_HOST_SRIOV	Along with ICP_SRIOV, this may be required to enable SRIOV for the host software installation.	Not defined	Defined when "Install SR-IOV Host Acceleration" is selected using the installer.sh installation script.
ICP_TRACE	Used to enable tracing capability for debug purposes. Once the acceleration driver is compiled with this option, all the Cryptography APIs will expose their parameters to	Not defined	

continued...

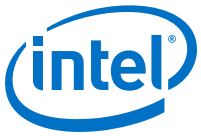
Symbol	Description	Default	Reference
	the console for user space applications OR to `/var/log/` messages in kernel space.		
`LAC_HW_PRECOMPUTES`	When defined, enables hardware for HMAC precomputes.	Not defined, therefore the driver uses software (dependency on OpenSSL and Linux Crypto API.	
`NB_MR_ROUNDS`	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., cpa_mux.ko will be built and will expose the Intel® QuickAssist Technology API. The drivers will not export symbols but will instead register with the cpa_mux.	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	
`ICP_DC_RETURN_COUNTERS_ON_ERROR`	Used to update the "consumed" and "produced" fields of the `CpaDcRqResults` structure even if an error occurs during compression or decompression operations.	Not defined	See implementation details provided under the final bullet of Intel QuickAssist Technology API Limitations on page 93

5.15 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)
```

6.0 Acceleration Driver Configuration File

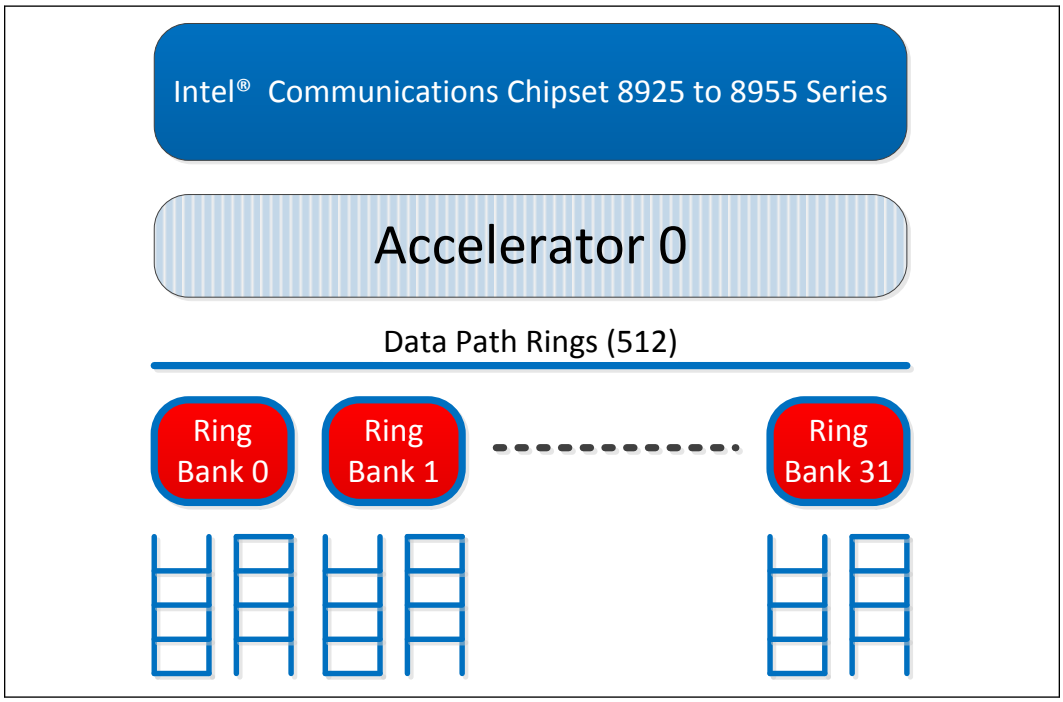
This chapter describes the configuration file(s) managed by the Acceleration Driver Framework (ADF) that allow customization of runtime operation. This configuration file(s) must be tuned to meet the performance needs of the target application.

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.

6.1 Configuration File Overview

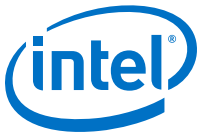
There is a single configuration file for each Intel® Communications Chipset 8925 to 8955 Series (PCH) device. Each ring bank has an interrupt that can be directed to a specific Intel® architecture core. Each ring bank has 16 rings (hardware assisted queues). This hierarchy is shown in the following figure.

Figure 13. 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 63.

  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:
  - 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 67.

A sample configuration file, targeted at a high-end IPsec box, is included in Sample Configuration File (V2) on page 76.

6.2 General Section

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

6.2.1 General Parameters

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

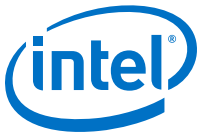
Table 4. General Parameters

Parameter	Description	Default	Range
WirelessEnabled	Enables use of optimized wireless service	0	0 or 1
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]).	cy;dc	cy, dc

continued...

Parameter	Description	Default	Range
			Note: 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 *Note:* In general, with this parameter set to 1, performance is expected to be better.
Firmware_MofPath	Path and Name of the Microcode (UCode) Object File (UOF) firmware.	dh895xcc/ mof_firmware.bin	
Firmware_MmpPath	Name of the Modular Math Processor (MMP) firmware.	dh895xcc/ mmp_firmware.bin	
CyNumConcurrentSymRequests	Specifies the number of cryptographic concurrent symmetric requests for cryptographic instances in general. *Note:* 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. *Note:* This parameter value can be overridden for a particular cryptographic instance if necessary.	64	64, 128, 256, 512, 1024, 2048 or 4096
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

continued...



Parameter	Description	Default	Range
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 117.	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	0 to 31
Note: "Default" denotes the value in the configuration file when shipped. *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.			

6.2.2 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/icp_dh895xcc_devX` directory, where X is the device number. For example, `/proc/icp_dh895xcc_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 46 for more information.

Table 5. Statistics Parameters

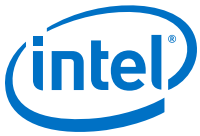
Parameter	Description	Default	Range
statsGeneral	Enables/disables statistics in general.	1	1 or 0
statsDh	Enables/disables statistics for the Diffie-Hellman algorithm.	1	1 or 0
statsDrbg	Enables/disables statistics for the Deterministic Random Bit Generator (DRBG).	1	1 or 0
statsDsa	Enables/disables 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
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".			

6.2.3 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 71), 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.
- 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.

6.3 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 32 and Logical Instances on page 19 for more information.

The address domains are in the following format:

- For the kernel address domain: `[KERNEL]`
- For user process address domains: `[xxxxx]`, where `xxxxx` 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 118 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)

6.3.1 [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 64; for exceptions, please see Configuration File Version 2 Differences on page 83.

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

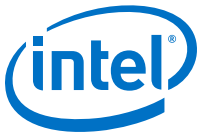
6.3.1.1 Cryptographic Logical Instance Parameters

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

Table 6. Cryptographic Logical Instance Parameters

Parameter	Description	Default	Range
CyXName	Specifies the name of cryptographic instance number X.	IPSec0	String (max. 64 characters)
CyXIsPolled	Specifies if cryptographic instance number X works in poll mode or IRQ mode.	0 for kernel space instances 1 for user space instances	0 (interrupt mode), 1 (poll mode)
CyXNumConcurrentSymRequests (optional)	Specifies the number of in-progress cryptographic concurrent symmetric requests (and responses) for cryptographic instance number X. *Note:* Overrides the default `CyNumConcurrentSymRequests` value in the General section for this specific instance. *Note:* In the configuration file, this parameter must be specified before the `CyXCoreAffinity` parameter. If it is not, the default value specified in `CyNumConcurrentSymRequests` 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. *Note:* Overrides the default `CyNumConcurrentAsymRequests` value in the General section for this specific instance.	N/A	64, 128, 256, 512, 1024, 2048 or 4096

continued...



Parameter	Description	Default	Range
	Note: In the configuration file, this parameter must be specified before the `CyXCoreAffinity` parameter. If it is not, the default value specified in `CyNumConcurrentAsymRequests` in the General section is used.		
CyXCoreAffinity	Specifies the core to which the instance should be affinitized.	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.			

6.3.1.2 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 64.

Parameter	Description	Default	Range
DcXName	Specifies the name of data compression instance number X.	IPComp0	String (max. 64 characters)
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	0 (interrupt mode), 1 (poll mode)
DcXNumConcurrentRequests (optional)	Specifies the number of data compression concurrent requests. Overrides the default `DcNumConcurrentRequests` value in the General section for this specific instance. *Note:* In the configuration file, this parameter must be specified before the `DcXCoreAffinity` parameter. If it is not, the default value specified in `DcNumConcurrentRequests` 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
Note: "Default" denotes the value in the configuration file when shipped.			

6.3.2 [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 67 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 102 for more information.

6.3.2.1 Dynamic Instance Configuration Example

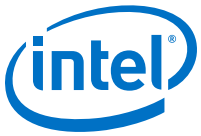
The following configuration file snippets demonstrate the reservation of instances for dynamic allocation. In a system that uses the two configuration files below, `icp_sal_userCyInstancesAlloc` can allocate up to 26 cryptographic (cy) instances and `icp_sal_userDcInstancesAlloc` can allocate up to 14 data compression (dc) instances. See Dynamic Instance Allocation Functions on page 102 for more information.

Taken from: `/etc/dh895xcc_qa_dev0.conf`

```
...

[DYN]
NumberCyInstances = 10
NumberDcInstances = 4

# Crypto - User instance DYN #0
Cy0Name = "DYN0"
Cy0IsPolled = 1

# List of core affinities
Cy0CoreAffinity = 0

# Crypto - User instance DYN #1
Cy1Name = "DYN1"
Cy1IsPolled = 1

# List of core affinities
Cy1CoreAffinity = 1

# Crypto - User instance DYN #2
Cy2Name = "DYN2"
Cy2IsPolled = 1

# 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
Dc1Name = "DCDYN1"
Dc1AcceleratorNumber = 1
Dc1IsPolled = 1
Dc1CoreAffinity = 1

...
```

Taken from: `/etc/dh895xcc_qa_dev1.conf`

```
...

[DYN]
NumberCyInstances = 16
```



```
NumberDcInstances = 10

...
```

6.3.3 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 7. User Process [xxxxx] Sections Parameters

Parameter	Description	Default	Range
NumProcesses	The number of user space processes with section name [xxxxx] that have access to this device. The maximum number of processes that can call icp_sal_userStartMultiProcess and be active at any one time. See icp_sal_userStartMultiProcess Usage on page 120 for more information. ***Caution:*** Resources are preallocated. If this parameter value is set too high, the driver fails to load.	1	For constraints, see Maximum Number of Process Calculations on page 71.
LimitDevAccess	Indicates if the user space processes in this section are limited to only access instances on this device.	0	0 (disabled, processes in this section can access multiple devices) or 1 (enabled, processes in this section can only access this device)
NumberCyInstances	Specifies the number of cryptographic instances. *Note:* Depends on the number of allocations to other services.	2	0 to 64
NumberDcInstances	Specifies the number of data compression instances. *Note:* Depends on the number of allocations to other services.	2	0 to 64
Note: "Default" denotes the value in the configuration file when shipped. *Note:* The order of `NumProcesses` and `LimitDevAccess` parameters is important. The `NumProcess` parameter must appear **before** the `LimitDevAccess` parameter in the section.			

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 67.

6.3.3.1 Maximum Number of Process Calculations

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

The value to which this parameters can be set is determined by a number of factors, most significantly, the number of cryptography instances and/or data compression instances in the processes. The total number of instances created by the driver is given by the expression:

```
(NumProcesses) x (NumberCyInstances)
```

There are 32 ring banks per Intel® Communications Chipset 8925 to 8955 Series device and a max of two cryptography instances and two compression instances per bank. This limits the maximum number of instances per device to 64 for cryptography and 64 for compression.

A further constraint is if interrupts are being used with user space processes. In this case, there is an interrupt vector per ring bank, and sharing of an interrupt vector and associated interrupt CSRs related to the bank between processes is not advised.

The following are examples that that illustrate the maximum number of processes possible with a device:

- All processes / instances in polling mode:

  `NumProcesses` = 64

  `NumCyInstances` = 1

  `NumDcInstances` = 1

- All processes / instances in interrupt mode:

  `NumProcesses` = 32

  `NumCyInstances` = 2

  `NumDcInstances` = 2

6.4 Configuring Multiple PCH Devices in a System

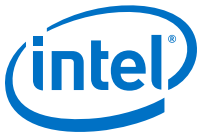
A platform may include more than one PCH device. Each device must have its own configuration file. The format and structure of the configuration file is exactly the same for all devices. Consequently, the configuration file for device 0, `dh895xcc_qa_dev0.conf`, can be cloned for use with other PCH devices.

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 `dh895xcc_qa_dev1.conf`, for device 2, change the file name to `dh895xcc_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 8086:0435
```



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

```
08:00.0 Co-processor: Intel Corporation Device 0435
09:00.0 Co-processor: Intel Corporation Device 0435
```

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
```

```
icp_dev0 - type=dh895xcc, inst_id=0, bsf=03:00:0, #accel=6, #engines=12, state=up
icp_dev1 - type=dh895xcc, inst_id=0, bsf=82:00:0, #accel=6, #engines=12, 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 45 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.

6.5 Configuring Multiple Processes on a Multiple-Device System

As an example, consider a system with two PCH devices 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 eight acceleration instances, four per device.

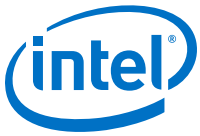
In this scenario, the user space section of the configuration files would look like the following.

For `dh895xcc_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

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

Cy0CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #1
Cy1Name = "IPSec1"
Cy1IsPolled = 1

Cy1CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #2
Cy2Name = "IPSec2"
Cy2IsPolled = 1

Cy2CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #3
Cy3Name = "IPSec3"
Cy3IsPolled = 1

Cy3CoreAffinity = 0 # Core affinity not used for polled instance
```

For `dh895xcc_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

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

Cy0CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #1
Cy1Name = "IPSec1"
Cy1IsPolled = 1

Cy1CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #2
Cy2Name = "IPSec2"
Cy2IsPolled = 1

Cy2CoreAffinity = 0 # Core affinity not used for polled instance
# Crypto - User instance #3
Cy3Name = "IPSec3"
Cy3IsPolled = 1

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.

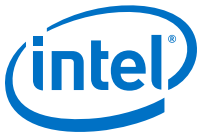
When a process calls the `icp_sal_userStartMultiProcess("IPSec", CPA_FALSE)` function, the driver locates the next available section of the form `IPSec_INT_<....>` (of which there is only one in total for this example) and assigns this section to the process. This gives the process access to the corresponding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

6.6 Sample Configuration File (V2)

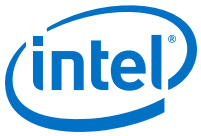
This following sample configuration file is provided in the software package.

```
############################################################################
#
# @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.1.3.0-90
############################################################################
############################################
# General Section
############################################

[GENERAL]
ServicesEnabled = cy0;cy1

# Use version 2 of the config file
ConfigVersion = 2

# Look Aside Cryptographic Configuration
cyHmacAuthMode = 1

# Firmware Location Configuration
Firmware_MofPath = mof_firmware_c2xxx.bin
Firmware_MmpPath = mmp_firmware_c2xxx.bin

#Default values for number of concurrent requests*/
CyNumConcurrentSymRequests = 512
CyNumConcurrentAsymRequests = 64

#Statistics, valid values: 1,0
statsGeneral = 1
statsDh = 1
statsDrbg = 1
statsDsa = 1
statsEcc = 1
statsKeyGen = 1
statsLn = 1
statsPrime = 1
statsRsa = 1
statsSym = 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.
#
```

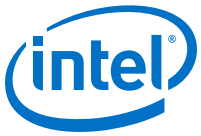
```
# The address domains are in the following format
# - For kernel address domains
#       [KERNEL]
# - For user process address domains
#   [xxxxx]
#   Where xxxxx may be any ascii value which uniquely identifies
#   the user mode process.
#   To allow the driver correctly configure the
#   logical 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 = 0-1
#               Cy<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 = 0,1
#             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 1
#
############################################################

##############################################
# Kernel Instances Section
##############################################
[KERNEL]
NumberCyInstances = 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

##############################################
# User Process Instance Section
##############################################
[SSL]
NumberCyInstances = 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

############################################
# Wireless Process Instance Section
############################################
[WIRELESS]
NumberCyInstances = 1
NumProcesses = 0

# Crypto - User instance #0
Cy0Name = "WIRELESS0"
Cy0IsPolled = 1
Cy0AcceleratorNumber = 0
# List of core affinities
Cy0CoreAffinity = 0
```

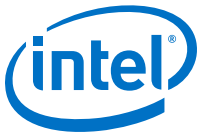
```
##############################################################################
#
# @par
# This file is provided under a dual BSD/GPLv2 license.  When using or
#   redistributing this file, you may do so under either license.
#
#   GPL LICENSE SUMMARY
#
#   Copyright(c) 2007-2013 Intel Corporation. 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: DH895xCC_ACCEL.L.0.5.0-80
############################################################################
############################################################
#
# This file is the configuration for a single dh895xcc_qa
# device.
#
# Each device has 32 independent banks.
#
# - Each bank can contain up to 2 crypto and/or up to 2 data
#   compression services.
#
# - The interrupt for each can be directed to a
#   specific core.
#
############################################################

################################################
# General Section
################################################

[GENERAL]
ServicesEnabled = cy;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 = dh895xcc/mof_firmware.bin
Firmware_MmpPath = dh895xcc/mmp_firmware.bin



#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

# Debug feature, if set to 1 it enables additional entries in /proc filesystem
ProcDebug = 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 needs to be disabled
# in the BIOS and Cy and/or Dc instances can be used in PF (Dom0)
SRIOV_Enabled = 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_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 response mode associated wth this logical instance (0
#   for IRQ or 1 for polled).
# - The core the instance is affinitized to (optional)
#
# The format of the logical instances are:
# - For crypto:
#               Cy<n>Name = "xxxx"
#               Cy<n>IsPolled = 0|1
#               Cy<n>CoreAffinity = 0-7
#
# - For Data Compression
#               Dc<n>Name = "xxxx"
#               Dc<n>IsPolled = 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 core affinity: for example
#             Cy0CoreAffinity = 0,2,4
# These comma-separated lists will allow multiple processes to use
# different accelerators and cores, and will wrap around the numbers
# in the list. In the above example, process 0 will have affinity 0,
# and process 1 will have affinity 2 etc.
#
#########################################################
```

```
############################################
# Kernel Instances Section
############################################
[KERNEL]
NumberCyInstances = 1
NumberDcInstances = 1

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

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

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

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

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

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

############################################
# User Process Instance Section
############################################
[SSL]
NumberCyInstances = 2
NumberDcInstances = 2
NumProcesses = 1
LimitDevAccess = 0

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

# Crypto - User instance #1
Cy1Name = "SSL1"
Cy1IsPolled = 1
# List of core affinities
Cy1CoreAffinity = 1

# Crypto - User instance #2
Cy2Name = "SSL2"
Cy2IsPolled = 1
# List of core affinities
Cy2CoreAffinity = 2

# Crypto - User instance #3
Cy3Name = "SSL3"
Cy3IsPolled = 1
# List of core affinities
Cy3CoreAffinity = 3
```



```
# Data Compression - User space  instance #0
Dc0Name = "UserDC0"
Dc0IsPolled = 1
Dc0CoreAffinity = 0

# Data Compression - User space  instance #1
Dc1Name = "UserDC1"
Dc1IsPolled = 1
Dc1CoreAffinity = 1
```

6.7 Compression Only SKU

In the case of the compression only SKU, only the DC service is supported on the device. This software support comes as part of the acceleration software package.

It is recommended to remove CY from the device config file(s) and set the NumberOfCyInstances to 0 for both kernel space and user space. For example:

[GENERAL]

ServicesEnabled = dc

No crypto requests will be supported. Any CY requests at the API level will return an error.

In the case of SR-IOV, the VF driver currently sees all capabilities regardless of SKU information. The VF driver currently does not have access to the registers which hold the SKU information. There are no threads mapped to the CY service when using this SKU. If CY is enabled in the VF devices config file, the behavior is undefined.

It is also recommended to explicitly set WirelessEnabled = 0 in the config file for this SKU. The wireless firmware does not support DC requests.

6.8 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.

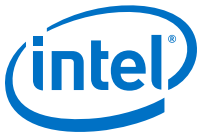
The following is a summary of the differences between the configuration file Version 2 and Version 1 file format:

- 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.
- 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 `CoreAffinity`=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 core 0 again, the fifth process uses core 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.



7.0 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 8925 to 8955 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 I/O component (based on the Intel® P55 Express Chipset) PCH.

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.

7.1 Terminology

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

- Threat Categories on page 85
- Attack Mechanism on page 85
- Attacker Privilege on page 86
- Deployment Models on page 86

7.1.1 Threat Categories

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

Table 8. System Threat Categories

Category	Nature of Threat and Examples
Exposure of Data	• Attacker reads data to which they should not have read access • Attacker reads cryptographic keys
Modification of Data	• Attacker overwrites data to which they should not have write access • Attacker overwrites cryptographic keys
Denial of Service	• 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

7.1.2 Attack Mechanism

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

Table 9. 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 *Note:* 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.

7.1.3 Attacker Privilege

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

Table 10. 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 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.

7.1.4 Deployment Models

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

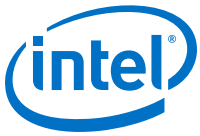


Table 11. Deployment Models

Deployment Model	Examples
System with no untrusted users	• Network security appliance • Server in data center
System with potentially untrusted users	• Server in data center

7.2 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.

7.2.1 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.

7.2.2 General Threats

General threats include the following:

- DMA on page 88
- Intentional Modification of IA Driver on page 88
- Modification of Intel QuickAssist Accelerator Firmware on page 88
- Malicious Application Code on page 89
- Contrived Packet Stream on page 89

7.2.2.1 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_mem`
- `icp_dev_mem_page`

The recommendation is that these files have the following permissions by default[1]:

```
# 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.

7.2.2.2 Intentional Modification of IA Driver

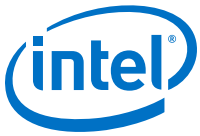
Threat: An attacker can potentially modify the IA driver to behave maliciously.

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.

7.2.2.3 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.

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



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 88 threat to limit access to this ring.

7.2.2.4 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, `dh895xcc_qa_dev0.conf`. Its default permissions are that it is readable and writeable only by root.

7.2.2.5 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 88.

7.2.2.6 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.

7.2.3 Threats Against the Cryptographic Service

Threats against the cryptographic service include:

- Reading and Writing of Cryptographic Keys on page 90
- Modification of Public Key Firmware on page 90
- Failure of the Entropy Source for the Random Number Generator on page 90
- Interference Among Users of the Random Number Service on page 91

7.2.3.1 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.

7.2.3.2 Modification of Public Key Firmware

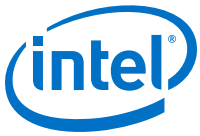
Background: In addition to the Intel® QuickAssist Accelerator firmware which is downloaded to the Acceleration Complex within the PCH by the driver at initialization time, there is a library of small public key firmware routines, one of which is downloaded to the device along with each request to perform 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.

7.2.3.3 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.

7.2.3.4 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.

7.2.4 Data Compression Service Threats

Threats against the Data Compression service include:

- Read/Write of Save/Restore Context on page 91
- Stateful Behavior on page 91
- Incomplete or Malformed Huffman Tree on page 92
- Contrived Packet Stream on page 92

7.2.4.1 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.

7.2.4.2 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.

7.2.4.3 Incomplete or Malformed Huffman Tree

Threat: An attacker who can run malicious code on the platform (see Malicious Application Code on page 89) 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.

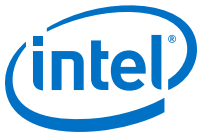
Mitigation: See the mitigation proposed in Malicious Application Code on page 89. Furthermore, the slice detects such incomplete or malformed Huffman trees and returns an error.

7.2.4.4 Contrived Packet Stream

Threat: Similar to the general attack mechanism described in Contrived Packet Stream on page 89, 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.



8.0 Supported APIs

The supported APIs are described in two categories:

- Intel® QuickAssist Technology APIs on page 93
- Additional APIs on page 101

8.1 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.

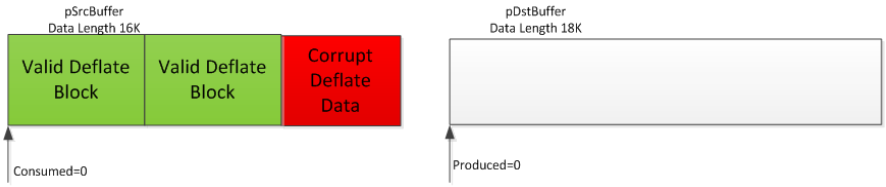
8.1.1 Intel® QuickAssist Technology API Limitations

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

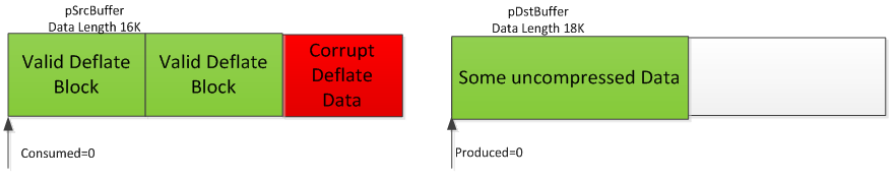
- For all services, the maximum size of a single *perform* request is 4 GB.
- For all services, data structures that contain data required by the Intel® QuickAssist Accelerator should be on a 64 Byte-aligned address to maximize performance. This alignment helps minimize latency when transferring data 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 <16 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, 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*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 (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*Total_Input_Byte/8) + 55 bytes needs to be supplied (even though the stored block output size might be less).
- For the data compression service, if an overflow occurs during stateless compression, the entire compression request must be resubmitted with a larger output buffer. In this case, the consumed value returned in the *cpaDcRqResults* structure will be zero.
- 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:



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

Table 12. Compression/Decompression Overflow Behavior

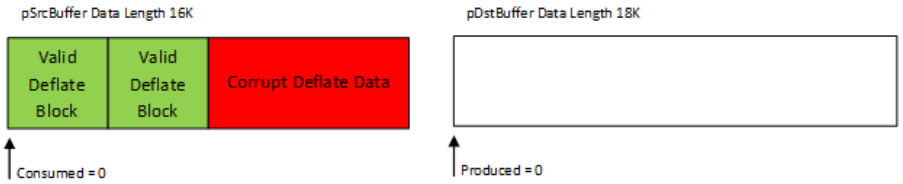
Stateful/ Stateless	Static/ Dynamic	Overflow	Input data consumed?	Valid data in output buffer?	Status Returned	Expected Application Behavior
Stateful	Both	Yes	Yes	Yes	-11	Submit next request - Input data pointer = next byte after consumed data of previous request - Output buffer: New output buffer (size does not matter)
Stateless	Both	Yes	No	No	-11	Re-submit the request - Input data pointer = same as for previous request - Output buffer = must be larger than previous request

Behavior when build flag ICP_DC_RETURN_COUNTERS_ON_ERROR is defined

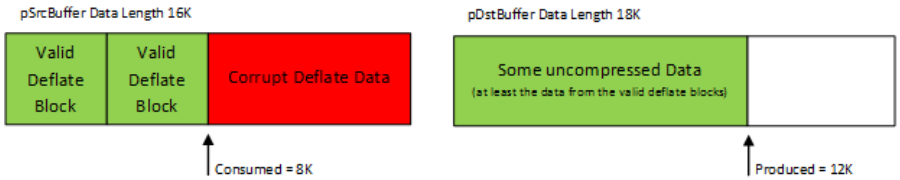
In some specialized applications, when a decompression soft error occurs, the application has no way of correcting the source of the error and resubmitting the request. The session will need to be invalidated and terminated. In this case it is more useful to the application to output the uncompressed data up to the point of soft error before terminating the session.

There is a compile time build flag (ICP_DC_RETURN_COUNTERS_ON_ERROR) to select this mode of operation. This is the behavior of decompression in case of soft error when this build flag is used.

If the following source and destination buffers were submitted to the Intel® QuickAssist Technology API:



The result would be:



It is important to note in this case:

— The checksum returned is not valid.

— The consumed value returned in the `CpaDcRqResults` structure is not reliable.

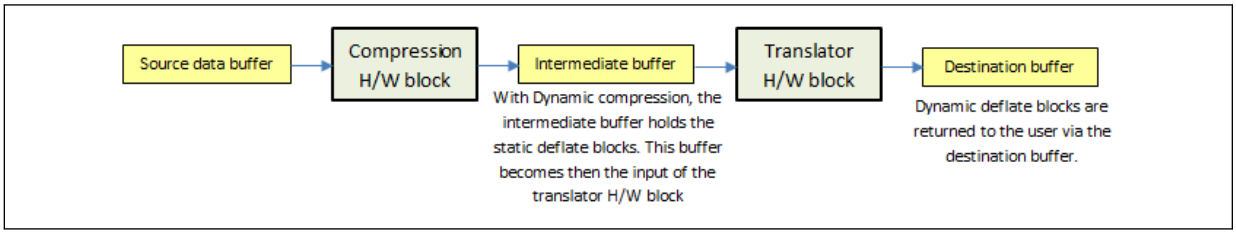
— No further requests can be submitted on this session.

— Overflow is treated as a soft error in the stateless case.

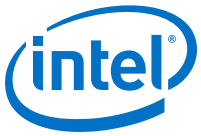
8.1.1.1 Dynamic Compression for Data Compression Service

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 14. 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.

8.1.1.2 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 though 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)
```

8.1.1.3 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:

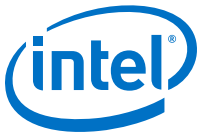
```
Static Produced data = ceil(9 * 111261 / 8) + 7
     = ceil (125168.625) + 7
     = 125169 + 7
Worst case Static Produced data = 125176 bytes
Memory required for destination buffer = ceil(9 * 111261 / 8) + 55
     = ceil (125168.625) + 55
     = 125169 + 7
     = 125169 + 55
     = 125224 bytes to 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 Intel QuickAssist Technology API Limitations on page 93.

8.1.2 Data Plane APIs Overview

The *Intel® QuickAssist Technology Cryptographic API Reference Manual* and the 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
```

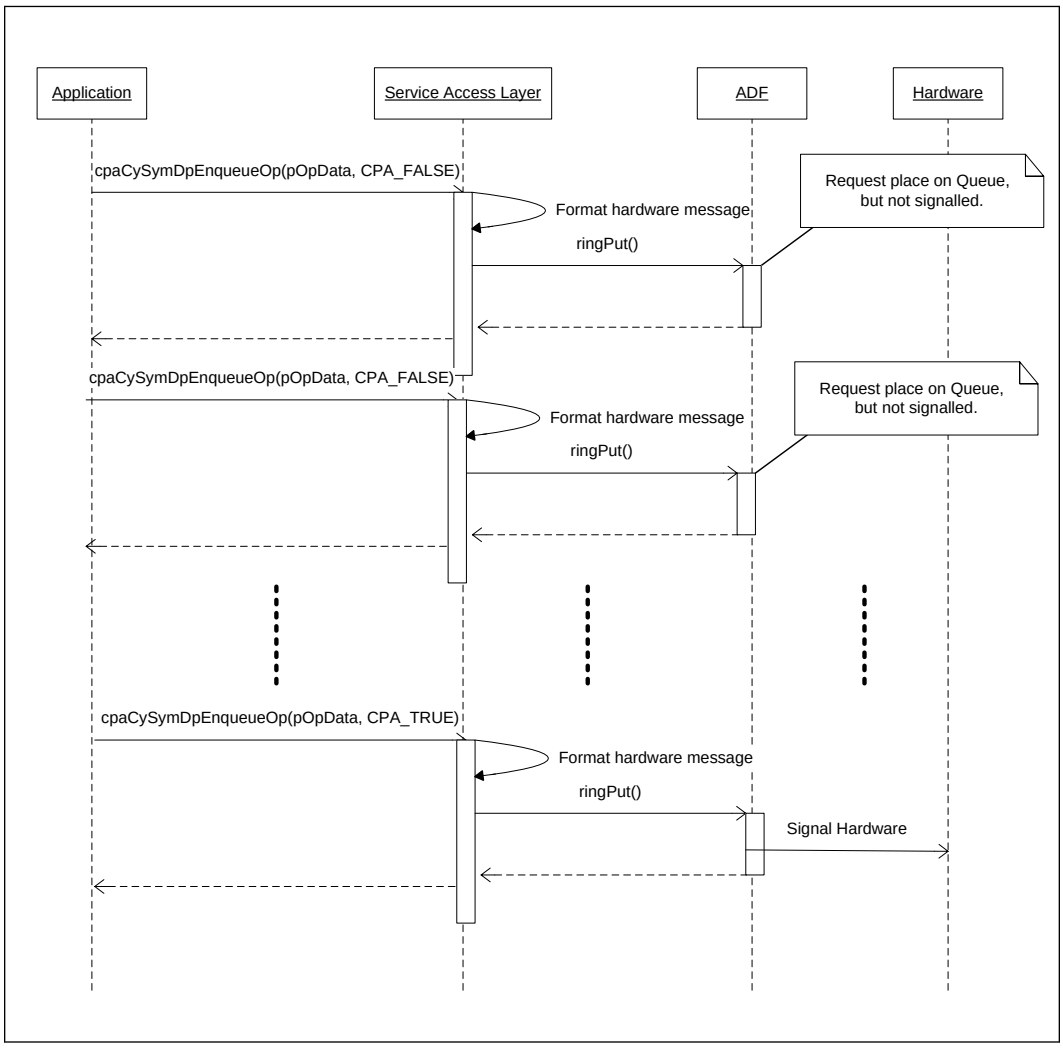
8.1.2.1 IA Cycle Count Reduction When Using Data Plane APIs

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 100).

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 15. Amortizing the Cost of an MMIO Across Multiple Requests



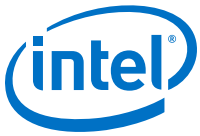
The Intel® QuickAssist Technology API returns a CPA_STATUS_RETRY when the ring becomes full.

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.

8.1.2.2 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 108) 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()`).
- The submitted requests are always placed on the high-priority ring.

8.1.2.3 Cryptographic API Descriptions

Full descriptions of the Intel® QuickAssist Technology APIs are contained in the *Intel® QuickAssist Technology Cryptographic 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 108.

8.2 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 102
- IOMMU Remapping Functions on page 106
- Polling Functions on page 108
- Random Number Generation Functions
- User Space Access Configuration Functions on page 118
- Version Information Function on page 122

8.2.1 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 reruns `./adf_ctrl d` and `./adf_ctrl u`.

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.

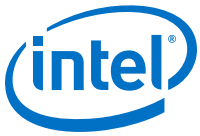
This can be useful when sharing instances among multiple applications at runtime. The maximum number of instances in a system is known in advance and it is possible to distribute them statically between applications using the configuration files. Once the driver is started, however, this cannot be changed. 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 103
- icp_sal_userDcGetAvailableNumDynInstances on page 103
- icp_sal_userCyInstancesAlloc on page 104
- icp_sal_userDcInstancesAlloc on page 104
- icp_sal_userCyFreeInstances on page 105
- icp_sal_userDcFreeInstances on page 105



8.2.1.1 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
CPA_STATUS_SUCCESS	Successfully retrieved the number of cryptographic instances available for dynamic allocation.
CPA_STATUS_FAIL	Indicates a failure.

8.2.1.2 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
CPA_STATUS_SUCCESS	Successfully retrieved the number of cryptographic instances available for dynamic allocation.
CPA_STATUS_FAIL	Indicates a failure.

8.2.1.3 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
CPA_STATUS_SUCCESS	Successfully allocated the sepecified number of cy instances.
CPA_STATUS_FAIL	Indicates a failure.

8.2.1.4 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);
```

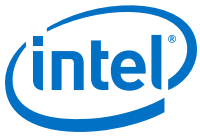
Parameters

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
CPA_STATUS_SUCCESS	Successfully allocated the specified number of dc instances.
CPA_STATUS_FAIL	Indicates a failure.



8.2.1.5 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
CPA_STATUS_SUCCESS	Successfully freed the specified number of cy instances.
CPA_STATUS_FAIL	Indicates a failure.

8.2.1.6 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
CPA_STATUS_SUCCESS	Successfully freed the specified number of dc instances.
CPA_STATUS_FAIL	Indicates a failure.

8.2.2 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 106
- icp_sal_iommu_map on page 106
- icp_sal_iommu_unmap on page 107

8.2.2.1 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.

8.2.2.2 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);
```

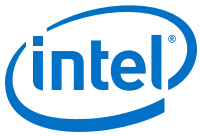
Parameters

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
CPA_STATUS_SUCCESS	Successful operation.
CPA_STATUS_FAIL	Indicates a failure.



8.2.2.3 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
CPA_STATUS_SUCCESS	Successful operation.
CPA_STATUS_FAIL	Indicates a failure.

8.2.2.4 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.

8.2.3 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_CyPollDpInstance

8.2.3.1 icp_sal_pollBank

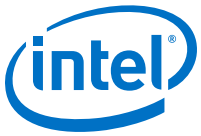
Poll all rings on the given accelerator on a given bank number to determine if any of the rings contain response messages from the Intel® QuickAssist Accelerator. The *response_quota* input parameter is per ring.

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 Intel® Communications Chipset 8925 to 8955 Series devices in the system.
bank_number	The number of the memory bank on the Intel® Communications Chipset 8925 to 8955 Series device that will be polled for response messages. The valid range is 0 to 31.
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
CPA_STATUS_SUCCESS	Successfully polled a ring with data.
CPA_STATUS_RETRY	There is no data on any ring on any bank or the banks are already being polled.
CPA_STATUS_FAIL	Indicates a failure.

8.2.3.2 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 Intel® Communications Chipset 8925 to 8955 Series 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
CPA_STATUS_SUCCESS	Successfully polled a ring with data.
CPA_STATUS_RETRY	There is no data on any ring on any bank or the banks are already being polled.
CPA_STATUS_FAIL	Indicates a failure.

8.2.3.3 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 141.

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
CPA_STATUS_SUCCESS	The function was successful.
CPA_STATUS_RETRY	There are no responses on the rings associated with the specified logical instance. *Note:* A ring is only polled if it contains data.
CPA_STATUS_FAIL	Indicates a failure.

8.2.3.4 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 141.

Syntax

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

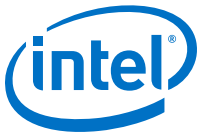
Parameters

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_DcPollInstance` function returns one of the following codes:

Code	Meaning
CPA_STATUS_SUCCESS	The function was successful.



Code	Meaning
CPA_STATUS_RETRY	There are no responses on the rings associated with the specified logical instance. *Note:* A ring is only polled if it contains data.
CPA_STATUS_FAIL	Indicates a failure.

8.2.3.5 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 100 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
CPA_STATUS_SUCCESS	The function was successful.
CPA_STATUS_RETRY	There are no responses on the rings associated with the specified logical instance.
CPA_STATUS_FAIL	Indicates a failure.

8.2.3.6 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 100 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
CPA_STATUS_SUCCESS	The function was successful.
CPA_STATUS_RETRY	There are no responses on the rings associated with the specified logical instance.
CPA_STATUS_FAIL	Indicates a failure.

8.2.4 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 available on all of the crypto instances.

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

Deterministic Random Bit Generator (DRBG) Support

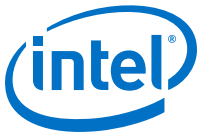
Implemented in software, DRBG processing takes some entropy as input and then performs Advanced Encryption Standard (AES) operations on the input using Intel® Communications Chipset 8925 to 8955 Series hardware.

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_drbgGetEnropyInputFuncRegister
- icp_sal_drbgGetNonceFuncRegister
- icp_sal_drbgHTGenerate
- icp_sal_drbgHTGetTestSessionSize
- icp_sal_drbgHTInstantiate
- icp_sal_drbgHTReseed
- icp_sal_drbgIsDFReqFuncRegister
- icp_sal_nrbgHealthTest

The `icp_sal_drbgGetEnropyInputFuncRegister`, `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.

8.2.4.1 icp_sal_drbgGetEnropyInputFuncRegister

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

Syntax

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

Parameters

func The function that the implementation may call to retrieve the DRGB 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/
```

8.2.4.2 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

8.2.4.3 icp_sal_drbgGetNonceFuncRegister

Allows the client to register a function that the implementation uses to retrieve the DRGB 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/
```

8.2.4.4 icp_sal_drbgHTGenerate

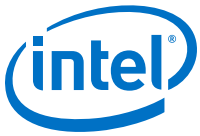
Tests the health of the Generate function as described in NIST SP 800-90, section 11.3.3.

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
CPA_STATUS_SUCCESS	Health tests passed.
CPA_STATUS_FAIL	Health tests failed.

8.2.4.5 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
CPA_STATUS_SUCCESS	Successfully retrieved the health test session size.
CPA_STATUS_FAIL	Indicates a failure.

8.2.4.6 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
CPA_STATUS_SUCCESS	Health tests passed.
CPA_STATUS_FAIL	Health tests failed.

8.2.4.7 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
CPA_STATUS_SUCCESS	Health tests passed.
CPA_STATUS_FAIL	Health tests failed.

8.2.4.8 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

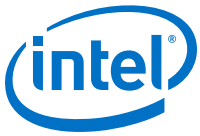
func	The function that the implementation may call to check if a derivation function is required.

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/
```

8.2.4.9 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
CPA_STATUS_SUCCESS	Health tests passed.
CPA_STATUS_RETRY	Resubmit the request.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in.
CPA_STATUS_RESOURCE	Error related to system resources.
CPA_STATUS_FAIL	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/
```

8.2.4.10 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".

A build option has been added to the acceleration driver to allow the `cpaCyDrbgSessionInit()` 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).

To enable this functionality:

1. Define "DRBG_POLL_AND_WAIT=1" before compilation of the acceleration driver. This can be done along with the other environmental variables (ICP_ROOT, ICP_ENV_DIR, and so on).
2. Proceed with driver build/installation.

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 63). 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.

8.2.5 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

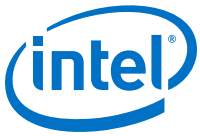
8.2.5.1 icp_sal_userStart

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
CPA_STATUS_SUCCESS	Successfully started user space access to the Intel® QuickAssist Accelerator.
CPA_STATUS_FAIL	Operation failed.

Notes

None

8.2.5.2 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 73 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
CPA_STATUS_SUCCESS	Successfully started user space access to the Intel® QuickAssist Accelerator as defined in the configuration file.
CPA_STATUS_FAIL	Operation failed.

8.2.5.2.1 icp_sal_userStartMultiProcess Usage

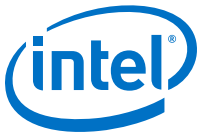
This topic describes a typical usage of the `icp_sal_userStartMultiProcess` function.

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 that the value configured by `NumProcesses` in the configuration file.
3. Each child process calls `icp_sal_userStartMultiProcess("SSL", CPA_TRUE)`. In 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.

8.2.5.3 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
CPA_STATUS_SUCCESS	Successfully stopped user space access to the Intel® QuickAssist Accelerator.
CPA_STATUS_FAIL	Operation failed.

Notes

None

8.2.6 User Space Heartbeat Functions

These functions allow the user space application to check the status of the firmware/hardware of the Intel® Communications Chipset 8925 to 8955 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 121
- icp_sal_check_all_devices on page 122

8.2.6.1 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
CPA_STATUS_SUCCESS	No error in operation.

Code	Meaning
CPA_STATUS_FAIL	Operation failed.

Notes

None

8.2.6.2 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
CPA_STATUS_SUCCESS	No error in operation.
CPA_STATUS_FAIL	Operation failed.

8.2.7 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`.

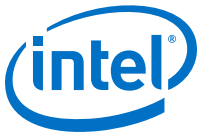
8.2.7.1 icp_sal_getDevVersionInfo

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
CPA_STATUS_SUCCESS	Operation finished successfully; version information retrieved.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed to the function.
CPA_STATUS_RESOURCE	System resource problem.
CPA_STATUS_FAIL	Operation failed.

8.2.8 PfVfComms Feature Functions

These APIs can only be called on a virtualized system in user space.

These functions allow messages to be sent between user-space applications on the Host and Guests. User messages are 14 bits of user-defined format and are targeted at a specific device and on the PF at a specific VF.

The transport channel is designed for infrequent usage, and is not suitable for carrying a heavy load. One CSR is available between the PF and each VF on each device; this CSR must be shared by users sending from both PF and VF side and by user and kernel space messages. It is reliable, i.e., the send_msg APIs will only return CPA_STATUS_SUCCESS if a message has been delivered to the driver on the other side; however, they can return CPA_STATUS_RETRY if the transport channel is in use. In this case, the API should be retried.

Retrieving messages is designed to be highly performant and non-blocking. To achieve this, the messages received by the kernel space driver are stored in memory mapped to each user-space process. Only the last message received on any channel is stored, so if the message buffer is not polled frequently enough, a message can be missed. The user-space driver keeps track of which messages have been retrieved so that the application is informed on the API call if a message has been missed. To make the interface non-blocking, this metadata is not locked, so the trade-off is that it is not multi-threaded, i.e., only one thread in each user-space process should use the "get" APIs.

All user-space PfVfComms function definitions are located in `$ICP_ROOT/quickassist/lookaside/access_layer/src/common/include/lac_common.h`

8.2.8.1 icp_get_pfvfcomms_status

This function returns CPA_TRUE if at least one message that has not been returned in a call to `icp_get_msg_from_pf` or `icp_get_msg_from_vf` is available on any channel.

Syntax

```
CpaStatus icp_get_pfvfcomms_status ( CpaBoolean *unreadMessage);
```

Parameters

unreadMessage Pointer to buffer to store status. Returns CPA_TRUE if at least one message is available on any channel which hasn't been returned in a call to `icp_get_msg_from_pf` or `icp_get_msg_from_vf`

Return Value

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

Code	Meaning
CPA_STATUS_SUCCESS	Successful operation.
CPA_STATUS_FAIL	Indicates a failure.

8.2.8.2 icp_send_msg_to_vf / icp_send_msg_to_pf

Send a message from vf to pf or vice versa.

Syntax

```
CpaStatus icp_send_msg_to_vf ( Cpa32U accelid, Cpa32U vfNum, Cpa32U message );

CpaStatus icp_send_msg_to_pf ( Cpa32U accelid, Cpa32U message );
```

Parameters

accelid The device number

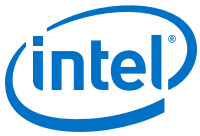
VfNum VF number. Range: 1-32

message 14 bit message. Range: 0-2^14-1 i.e. bits 14-31 will be masked off and only bits 0-13 passed across the comms channel. The 14 bit message can be in any user-defined format.

Return Value

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

Code	Meaning
CPA_STATUS_SUCCESS	Successful operation.
CPA_STATUS_FAIL	Indicates a failure.
CPA_STATUS_RETRY	Transport channel is busy, try again later.



Code	Meaning
CPA_STATUS_UNSUPPORTED	Returned if API called on a non-virtualized system
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in API

8.2.8.3 icp_get_msg_from_vf / icp_get_msg_from_pf

Get message from vf or pf.

Syntax

```
CpaStatus icp_get_msg_from_vf ( Cpa32U accelid, Cpa32U vfNum, Cpa32U * message,
Cpa32U * messageCounter);

CpaStatus icp_get_msg_from_pf ( Cpa32U accelid, Cpa32U * message, Cpa32U *
messageCounter);
```

Parameters

accelid	The device number
vfNum	VF number. Range: 1-32
message	Pointer to buffer to store bit message. The message will be returned in the bottom 14 bits.
messageCounter	pointer to buffer to store the number of messages received on this channel since API last called. • 0 => No new message • 1 => One message available • n (>1) => Last message available, but missed n-1 messages. As only the last message per device (and on the PF per VF) is stored a message could be missed if the API is not called often enough. This value allows the application to detect this.

Return Value

The `icp_get_msg_from_vf` or `icp_get_msg_from_pf` function returns one of the following codes:

Code	Meaning
CPA_STATUS_SUCCESS	Successful operation.
CPA_STATUS_FAIL	Indicates a failure.
CPA_STATUS_UNSUPPORTED	Returned if API called on a non-virtualized system.
CPA_STATUS_INVALID_PARAM	Invalid parameter passed in API.

8.2.9 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 49 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`.

8.2.9.1 icp_reset_device

Resets the device.

Syntax

```
CpaStatus icp_reset_device ( Cpa32U accelid);
```

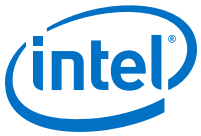
Parameters

accelid	The device number.

Return Value

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

Code	Meaning
CPA_STATUS_SUCCESS	Successful operation.
CPA_STATUS_FAIL	Indicates a failure.



Part 4: Applications and Usage Models

9.0 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.

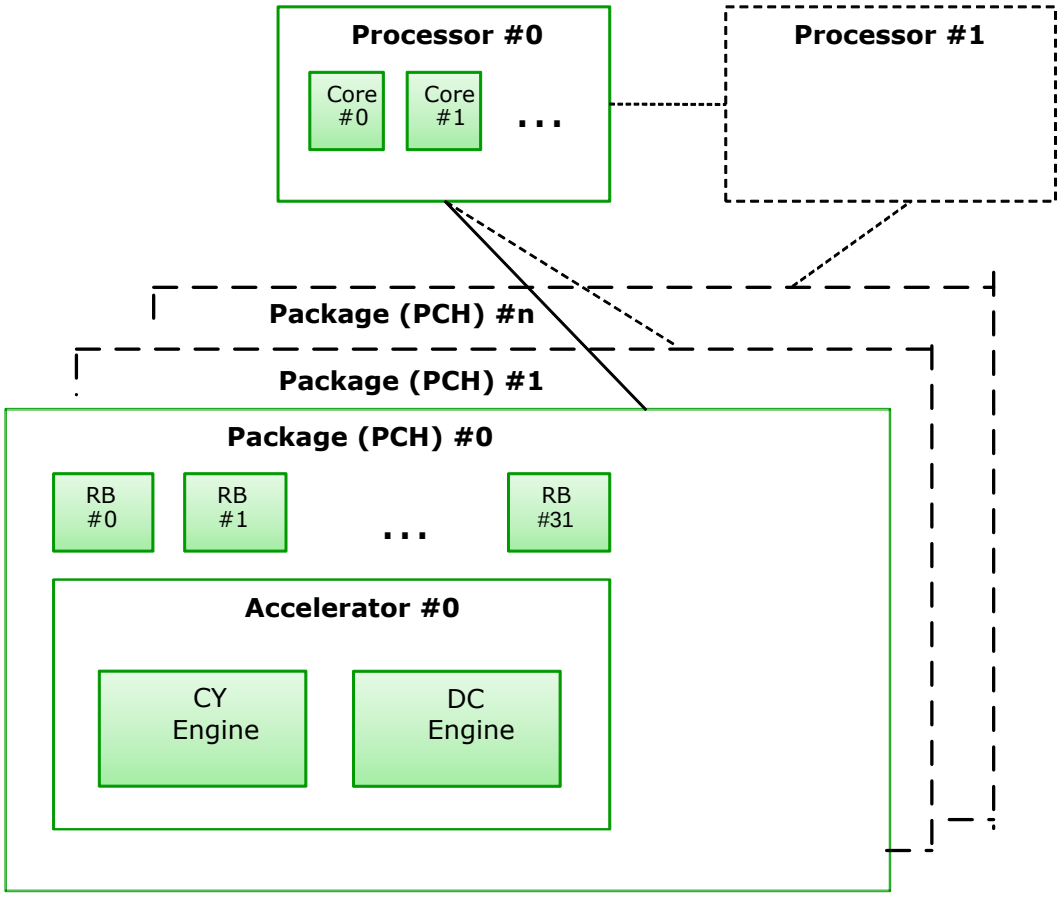
9.1 Mapping Service Instances to Hardware Accelerators on the PCH

On the platform(s) described in this manual, a processor can be connected to one or more Intel® Communications Chipset 8925 to 8955 Series (PCH) devices. Each PCH device contains one logical accelerator from a software perspective. Physically, each device contains multiple accelerators which are abstracted behind a load balancing hardware component. All requests sent to the one logical accelerator will be load balanced automatically across the physical accelerators within a PCH device. This is a key difference between previous generation 89xx devices.

A set of 32 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 16. Processor and PCH Device Components

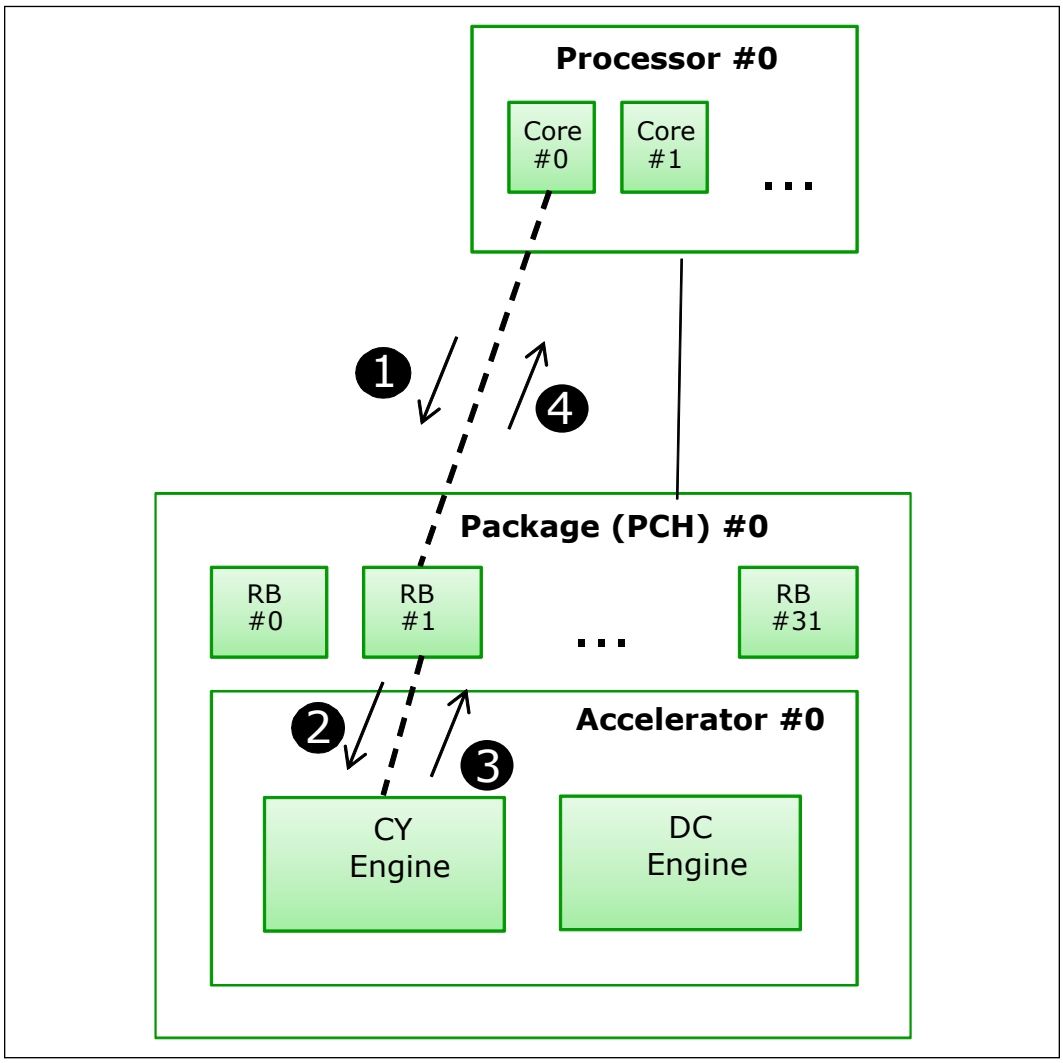


9.1.1 Processor and PCH Device Communication

An acceleration service uses different rings for request and response messages. 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 17. Processor and PCH Device Communication



9.1.2 Service Instances and Interaction with the Hardware

A ring bank supports two crypto instances and two compression instances.A 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 (aka bulk) cryptography requests/responses
- TRNG requests/responses
- Public key cryptography requests/responses

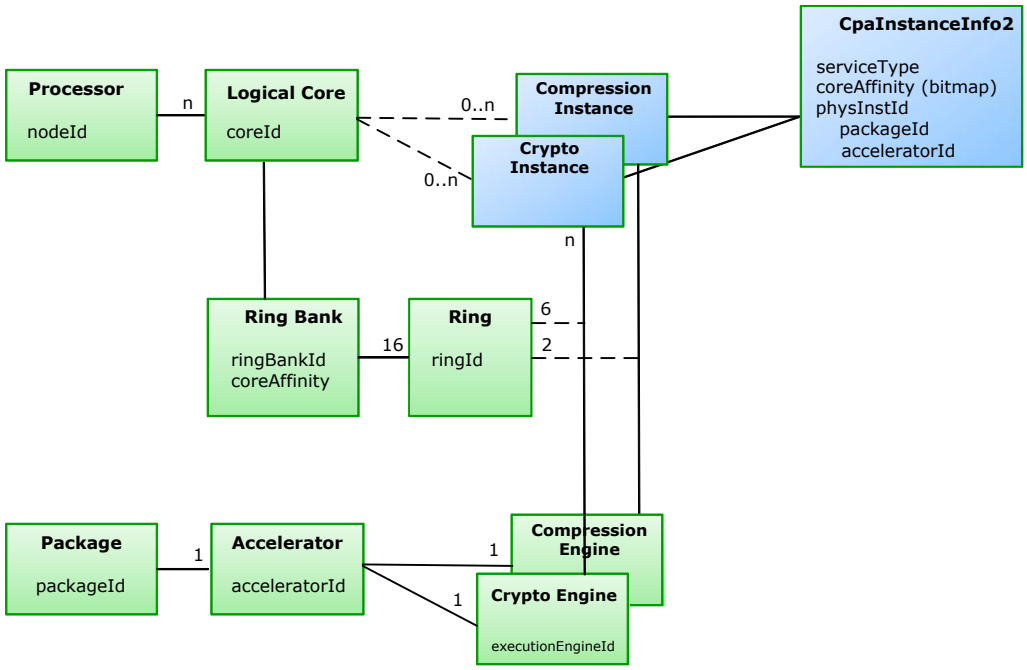
The key attributes of a service instance are given in the following table.

Table 13. Service Instance Attributes

Member	Sub-field	Description
coreAffinity	N/A	Identifies the core(s) to which interrupts (if enabled) are affinitized (Bitmap)
isPolled	N/A	When delivering responses to the client, specifies whether: • Interrupts are enabled (0) • Polling is used (1)

The following figure shows how the attributes relate to hardware components.

Figure 18. Service Instance Attributes and Hardware Components



9.1.3 Service Instance Configuration

The configuration of a service instance is done in the configuration file.

The following figure shows an example extract of the relevant section in the configuration file.

Figure 19. Service Instance Configuration

```
##############################################
# User Space Instances Section
##############################################
[proc0] ❶
NumberCyInstances = 1
NumberDcInstances = 0

# Crypto - user space instance #0
Cy0Name = "proc0_0" ❷
Cy0IsPolled = 1 ❸
Cy0CoreAffinity = 0 ❹
```

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. Specifies a name for the instance.
3. Specifies that the instance is using polling.
4. Specifies the core affinity for the instance.

9.1.4 Guidelines for Using Multiple Intel® QuickAssist Instances for Load Balancing in Cryptography Applications

The application is responsible for load balancing/spreading requests across PCH devices. Load balancing across the Intel® QuickAssist Technology accelerators within the PCH device is performed by hardware.

Maximum performance from the hardware can be obtained from either of the following service instance configurations:

- A single service instance
- Multiple service instances

Note: Depending on the specific design of an application that uses the hardware, using multiple service instances may be required to get full performance.

When the PCH device has more capacity than required by an logical core, each logical core can be assigned a different service instance. The load is balanced by spreading the traffic across logical cores. When the capacity of the PCH device can be handled by a single logical core, a single service instance can used and assigned to this logical core.

9.2 Cryptography Applications

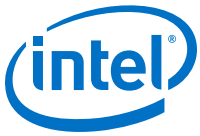
Cryptography applications supported by the platforms described in this manual include, but are not limited to:

- 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 132 for more information.
- Encrypted Storage. See Encrypted Storage on page 133 for more information.
- Web Proxy Appliances. See Web Proxy Appliances on page 134.

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.

9.2.1 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 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).

9.2.2 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 8925 to 8955 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.

9.2.3 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 8925 to 8955 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.

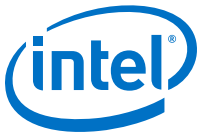
9.3 Data Compression Applications

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.

9.3.1 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 8925 to 8955 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.

9.3.2 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.

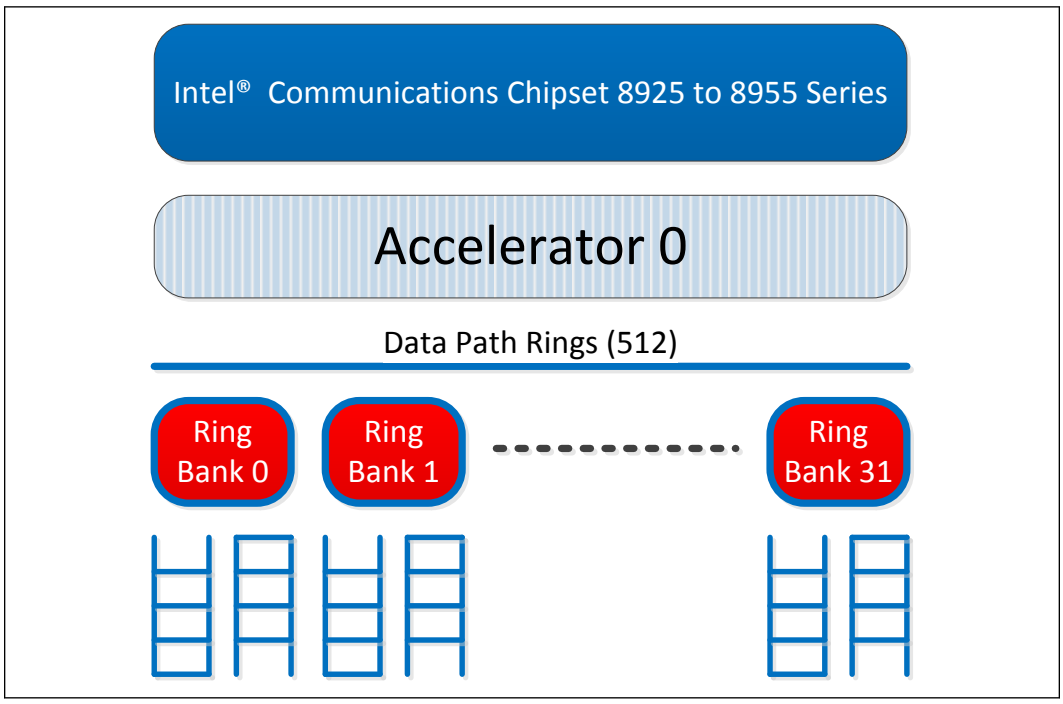
This chapter describes the configuration file(s) managed by the Acceleration Driver Framework (ADF) that allow customization of runtime operation. This configuration file(s) must be tuned to meet the performance needs of the target application.

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 8925 to 8955 Series (PCH) device. The configuration file contains one accelerator subsection. The accelerator has 32 independent ring banks (see the following figure).

Figure 20. Ring Banks



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 137.
- **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 [Accelerator0] Section on page 137.
- **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 139.

A sample configuration file, targeted at a high-end IPsec box, is included in Sample Configuration File (V1) on page 142.

A.2 General Section

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

A.2.1 General Parameters

Please see Table 4 on page 63

A.2.2 Statistics Parameters

Please see Table 5 on page 66

A.3 [Accelerator0] Section

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 14. 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 31.	1	0 or 1
BankXInterruptCoalescingTimerNs	Specifies the coalescing time, in nanoseconds (ns), for ring bank X, where X is in the range 0 to 31. *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.			

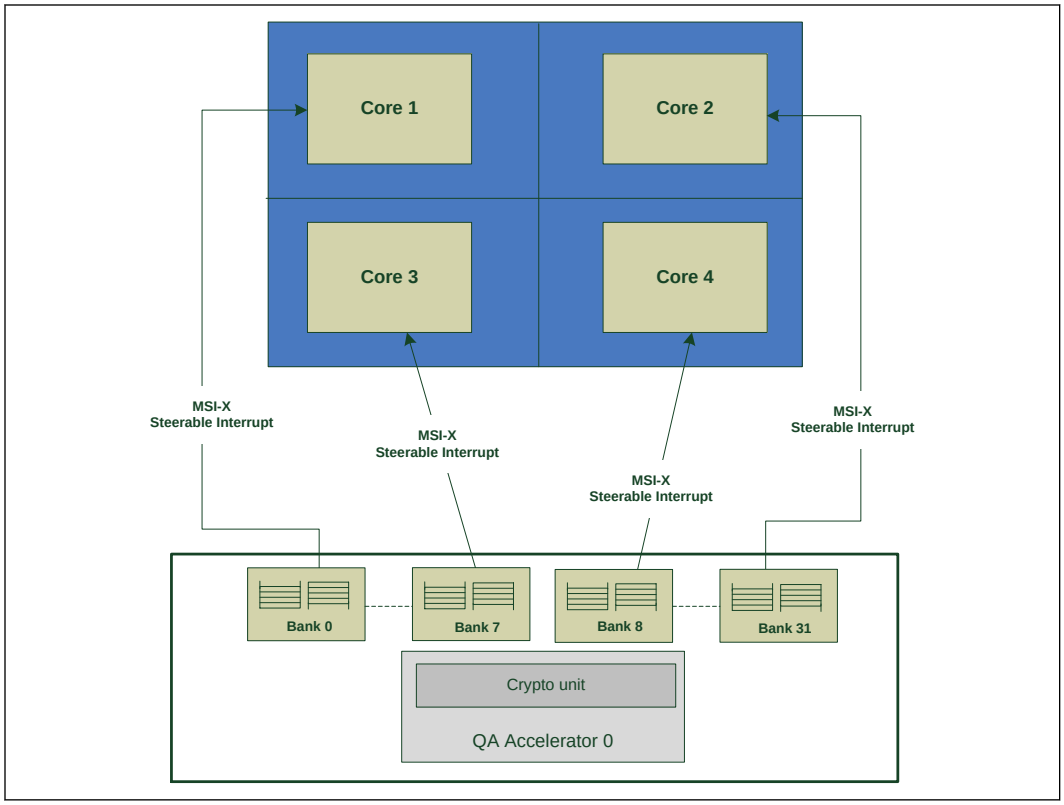
A.3.2 Affinity Parameters

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 32 ring banks (0 to 31). If the OS supports MSI-X interrupts, each ring bank has a steerable MSI-X interrupt that may be affinitized to a particular node/core as shown in the following figure.

Figure 21. Ring Bank Affinity to Core for MSI-X Interrupts



For each accelerator, the ring bank parameters in the following table can be configured.

Table 15. Ring Bank Affinity Parameters

Parameter	Description	Default	Range
BankXCoreIDAffinity	Defines core affinity for ring bank X, where X is in the range 0 to 31.	0	0 to cpumax-1 *Note:* cpumax is the number of CPUs in the system.
Note: "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 32 and Logical Instances on page 19 for more information.

The address domains are in the following format:

- For the kernel address domain: `[KERNEL]`
- For user process address domains: `[xxxxx]`, where `xxxxx` 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 118, passing the xxxxx string during process initialization. When the user space process is finished, it must call the function icp_sal_userStop on page 120 to free resources. See User Space Access Configuration Functions on page 118 for more information.

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

- The name of the 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 rings for receiving and the rings 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 = 512 (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 64.

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

A.4.1.1 User Process Instance [xxxxx] Sections

For information about the number and type of user process instances, please see Table 7 on page 71

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 139.

A.4.1.2 Cryptographic Logical Instance Parameters

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

Table 16. 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)
CyXBankNumber	Specifies the bank number of the cryptographic instance number X.	0 for kernel space instances 1 for user space instances	0 to 31
CyXIsPolled	Specifies if cryptographic instance number X works in poll mode or IRQ mode.	0 for kernel space instances 1 for user space instances	0 (interrupt mode), 1 (poll mode)
CyXNumConcurrentSymRequests	Specifies the number of cryptographic concurrent symetric 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.	0	0 or 1
CyXRingAsymRx	Specifies the asymmetric response ring number for cryptographic instance number X.	8	Must be Tx+8, i.e., 8 or 9
CyXRingSymTx	Specifies the symmetric request ring number for cryptographic instance number X messages.	2	2 or 3
CyXRingSymRx	Specifies the symmetric response ring number for cryptographic instance number X for messages.	10	10 or 11
CyXRingNrbgTx	Specifies the NRBG transmit ring number for cryptographic instance number X.	4	4 or 5
CyXRingNrbgRx	Specifies the NRBG response ring number for cryptographic instance number X.	12	Must be Tx+8, i.e., 12 or 13
Note: "Default" denotes the value in the configuration file when shipped.			

A.4.1.3 Data Compression Logical Instance Parameters

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

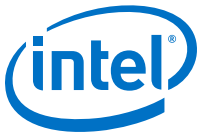
Note: The maximum number of data compression instances supported is 64.

Parameter	Description	Default	Range
DcXName	Specifies the name of data compression instance number X.	IPComp0	String (max. 64 characters)
DcXBankNumber	Specifies the bank number of data compression instance number X.	0 for kernel space instances 1 for user space instances	0 to 8
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	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.	6	6 or 7
DcXRingRx	Specifies the response ring number for data compression instance number X.	14	Must be Rx+8, i.e., 14 or 15
Note: "Default" denotes the value in the configuration file when shipped.			

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
```

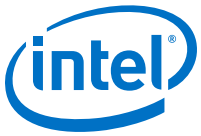


```
#
#   BSD LICENSE
#
#   Copyright(c) 2007-2013 Intel Corporation. All rights reserved.
#   All rights reserved.
#
#   Redistribution and use in source and binary forms, with or without
#   modification, are permitted provided that the following conditions
#   are met:
#
#     * Redistributions of 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: DH895xCC_ACCEL.L.0.5.0-80
############################################################################
############################################################################
#
# This file is the configuration for a single dh895xcc_qa
# device.
#
# Each device has 32 independent banks.
#
# - Each bank can contain up to 2 crypto and/or up to 2 data
#   compression services.
#
# - The interrupt for each can be directed to a
#   specific core.
#
############################################################################

##############################################
# General Section
##############################################

[GENERAL]
ServicesEnabled = cy;dc

# Look Aside Cryptographic Configuration
cyHmacAuthMode = 1

# Wireless Enabled
WirelessEnabled = 0

# Firmware Location Configuration
Firmware_MofPath = dh895xcc/mof_firmware.bin
Firmware_MmpPath = dh895xcc/mmp_firmware.bin

# Default values for number of concurrent requests
CyNumConcurrentSymRequests = 512
CyNumConcurrentAsymRequests = 64
```

```
# Default number of DC concurrent requests.
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

# Debug feature, if set to 1 it enables additional entries in /proc filesystem
ProcDebug = 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 needs to be disabled
# in the BIOS and Cy and/or Dc instances can be used in PF (Dom0)
SRIOV_Enabled = 0

##############################################################################
#
# Hardware Access Bank Configuration
# Each device has 32 banks (0-31)
# If the OS supports MSI-X, each bank has an
# steerable MSI-x interrupt which may be
# affinitized to a particular core.
#
# There is only one logical accelerator:
#   [Accelerator0]
#
# Items configurable per bank are:
# - Interrupt Coalescing Enabled (MSI-x interrupts)
# - The time in nano seconds before a coalesced interrupt is asserted
# - The core to steer interrupts for this bank to
# - Interrupt Coalescing based on the number of responses
# - Latest time after a response has been put on the Rx ring
#   when (IAAP) autopushed metadata is available. For polled rings,
#   the poll period should be at least twice this
#
# The format of the bank configurations are:
#               Bank<n>InterruptCoalescingEnabled = "xxxx"
#               Bank<n>InterruptCoalescingTimerNs = "xxxx"
#               Bank<n>CoreIDAffinity = "xxxx"
#               Bank<n>InterruptCoalescingNumResponses = "xxxx"
#
# Where:
#       - n is the number of the bank starting at 0.
#
##############################################################################

[Accelerator0]
Bank0InterruptCoalescingEnabled = 1
Bank0InterruptCoalescingTimerNs = 10000
Bank0CoreIDAffinity = 0
Bank0InterruptCoalescingNumResponses = 0

Bank1InterruptCoalescingEnabled = 1
Bank1InterruptCoalescingTimerNs = 10000
Bank1CoreIDAffinity = 1
Bank1InterruptCoalescingNumResponses = 0

Bank2InterruptCoalescingEnabled = 1
Bank2InterruptCoalescingTimerNs = 10000
Bank2CoreIDAffinity = 2
```

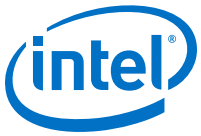
```
Bank2InterruptCoalescingNumResponses = 0

Bank3InterruptCoalescingEnabled = 1
Bank3InterruptCoalescingTimerNs = 10000
Bank3CoreIDAffinity = 3
Bank3InterruptCoalescingNumResponses = 0

Bank4InterruptCoalescingEnabled = 1
Bank4InterruptCoalescingTimerNs = 10000
Bank4CoreIDAffinity = 4
Bank4InterruptCoalescingNumResponses = 0

Bank5InterruptCoalescingEnabled = 1
Bank5InterruptCoalescingTimerNs = 10000
Bank5CoreIDAffinity = 5
Bank5InterruptCoalescingNumResponses = 0

Bank6InterruptCoalescingEnabled = 1
Bank6InterruptCoalescingTimerNs = 10000
Bank6CoreIDAffinity = 6
Bank6InterruptCoalescingNumResponses = 0

Bank7InterruptCoalescingEnabled = 1
Bank7InterruptCoalescingTimerNs = 10000
Bank7CoreIDAffinity = 7
Bank7InterruptCoalescingNumResponses = 0

Bank8InterruptCoalescingEnabled = 1
Bank8InterruptCoalescingTimerNs = 10000
Bank8CoreIDAffinity = 8
Bank8InterruptCoalescingNumResponses = 0

Bank9InterruptCoalescingEnabled = 1
Bank9InterruptCoalescingTimerNs = 10000
Bank9CoreIDAffinity = 9
Bank9InterruptCoalescingNumResponses = 0

Bank10InterruptCoalescingEnabled = 1
Bank10InterruptCoalescingTimerNs = 10000
Bank10CoreIDAffinity = 10
Bank10InterruptCoalescingNumResponses = 0

Bank11InterruptCoalescingEnabled = 1
Bank11InterruptCoalescingTimerNs = 10000
Bank11CoreIDAffinity = 11
Bank11InterruptCoalescingNumResponses = 0

Bank12InterruptCoalescingEnabled = 1
Bank12InterruptCoalescingTimerNs = 10000
Bank12CoreIDAffinity = 12
Bank12InterruptCoalescingNumResponses = 0

Bank13InterruptCoalescingEnabled = 1
Bank13InterruptCoalescingTimerNs = 10000
Bank13CoreIDAffinity = 13
Bank13InterruptCoalescingNumResponses = 0

Bank14InterruptCoalescingEnabled = 1
Bank14InterruptCoalescingTimerNs = 10000
Bank14CoreIDAffinity = 14
Bank14InterruptCoalescingNumResponses = 0

Bank15InterruptCoalescingEnabled = 1
Bank15InterruptCoalescingTimerNs = 10000
Bank15CoreIDAffinity = 15
Bank15InterruptCoalescingNumResponses = 0

Bank16InterruptCoalescingEnabled = 1
Bank16InterruptCoalescingTimerNs = 10000
Bank16CoreIDAffinity = 0
Bank16InterruptCoalescingNumResponses = 0
```

```
Bank17InterruptCoalescingEnabled = 1
Bank17InterruptCoalescingTimerNs = 10000
Bank17CoreIDAffinity = 1
Bank17InterruptCoalescingNumResponses = 0

Bank18InterruptCoalescingEnabled = 1
Bank18InterruptCoalescingTimerNs = 10000
Bank18CoreIDAffinity = 2
Bank18InterruptCoalescingNumResponses = 0

Bank19InterruptCoalescingEnabled = 1
Bank19InterruptCoalescingTimerNs = 10000
Bank19CoreIDAffinity = 3
Bank19InterruptCoalescingNumResponses = 0

Bank20InterruptCoalescingEnabled = 1
Bank20InterruptCoalescingTimerNs = 10000
Bank20CoreIDAffinity = 4
Bank20InterruptCoalescingNumResponses = 0

Bank21InterruptCoalescingEnabled = 1
Bank21InterruptCoalescingTimerNs = 10000
Bank21CoreIDAffinity = 5
Bank21InterruptCoalescingNumResponses = 0

Bank22InterruptCoalescingEnabled = 1
Bank22InterruptCoalescingTimerNs = 10000
Bank22CoreIDAffinity = 6
Bank22InterruptCoalescingNumResponses = 0

Bank23InterruptCoalescingEnabled = 1
Bank23InterruptCoalescingTimerNs = 10000
Bank23CoreIDAffinity = 7
Bank23InterruptCoalescingNumResponses = 0

Bank24InterruptCoalescingEnabled = 1
Bank24InterruptCoalescingTimerNs = 10000
Bank24CoreIDAffinity = 8
Bank24InterruptCoalescingNumResponses = 0

Bank25InterruptCoalescingEnabled = 1
Bank25InterruptCoalescingTimerNs = 10000
Bank25CoreIDAffinity = 9
Bank25InterruptCoalescingNumResponses = 0

Bank26InterruptCoalescingEnabled = 1
Bank26InterruptCoalescingTimerNs = 10000
Bank26CoreIDAffinity = 10
Bank26InterruptCoalescingNumResponses = 0

Bank27InterruptCoalescingEnabled = 1
Bank27InterruptCoalescingTimerNs = 10000
Bank27CoreIDAffinity = 11
Bank27InterruptCoalescingNumResponses = 0

Bank28InterruptCoalescingEnabled = 1
Bank28InterruptCoalescingTimerNs = 10000
Bank28CoreIDAffinity = 12
Bank28InterruptCoalescingNumResponses = 0

Bank29InterruptCoalescingEnabled = 1
Bank29InterruptCoalescingTimerNs = 10000
Bank29CoreIDAffinity = 13
Bank29InterruptCoalescingNumResponses = 0

Bank30InterruptCoalescingEnabled = 1
Bank30InterruptCoalescingTimerNs = 10000
Bank30CoreIDAffinity = 14
Bank30InterruptCoalescingNumResponses = 0
```



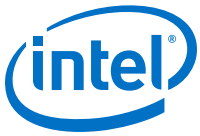
```
Bank31InterruptCoalescingEnabled = 1
Bank31InterruptCoalescingTimerNs = 10000
Bank31CoreIDAffinity = 15
Bank31InterruptCoalescingNumResponses = 0


##########################################################
#
# Logical Instances Section
# A logical instance allows each address domain
# (kernel space and individual user space processes)
# to be allocated to a bank and to define the
# behavior of that bank.
# - N.B. A single bank cannot be shared between two
#   address domains.
#
# 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 finished 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 bank associated with this logical
#   instance.
# - The response mode associated wth this logical instance (0
#   for IRQ or 1 for polled).
# - The number of concurrent requests supported. 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.
# - The Ring number. Rx ring number = Tx ring number + 8
#
# The format of the logical instances are:
# - For crypto:
#               Cy<n>Name = "xxxx"
#               Cy<n>BankNumber = 0-31
#               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|1
#               Cy<n>RingAsymRx = 8|9
#               Cy<n>RingSymTx = 2|3
#               Cy<n>RingSymRx = 10|11
#               Cy<n>RingNrbgTx = 4|5
#               Cy<n>RingNrbgRx = 12|13
#
# - For Data Compression
#               Dc<n>Name = "xxxx"
#               Dc<n>BankNumber = 0-31
#               Dc<n>IsPolled = 0|1
#               Dc<n>NumConcurrentRequests = 64|128|256|512|1024|2048|4096
#               Dc<n>RingTx = 6|7
#               Dc<n>RingRx = 14|15
#
# 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"
Cy0BankNumber = 0
Cy0IsPolled = 0
Cy0NumConcurrentSymRequests = 512
Cy0NumConcurrentAsymRequests = 64
Cy0RingAsymTx = 0
Cy0RingAsymRx = 8
Cy0RingSymTx = 2
Cy0RingSymRx = 10
Cy0RingNrbgTx = 4
Cy0RingNrbgRx = 12

# Crypto - Kernel instance #1
Cy1Name = "IPSec1"
Cy1BankNumber = 1
Cy1IsPolled = 0
Cy1NumConcurrentSymRequests = 512
Cy1NumConcurrentAsymRequests = 64
Cy1RingAsymTx = 0
Cy1RingAsymRx = 8
Cy1RingSymTx = 2
Cy1RingSymRx = 10
Cy1RingNrbgTx = 4
Cy1RingNrbgRx = 12

# Crypto - Kernel instance #2

Cy2Name = "IPSec2"
Cy2BankNumber = 0
Cy2IsPolled = 0
Cy2NumConcurrentSymRequests = 512
Cy2NumConcurrentAsymRequests = 64
Cy2RingAsymTx = 1
Cy2RingAsymRx = 9
Cy2RingSymTx = 3
Cy2RingSymRx = 11
Cy2RingNrbgTx = 5
Cy2RingNrbgRx = 13

# Crypto - Kernel instance #3
Cy3Name = "IPSec3"
Cy3BankNumber = 1
Cy3IsPolled = 0
Cy3NumConcurrentSymRequests = 512
Cy3NumConcurrentAsymRequests = 64
Cy3RingAsymTx = 1
Cy3RingAsymRx = 9
Cy3RingSymTx = 3
Cy3RingSymRx = 11
Cy3RingNrbgTx = 5
Cy3RingNrbgRx = 13


# Data Compression - Kernel instance #0
Dc0Name = "IPComp0"
Dc0IsPolled = 0
Dc0NumConcurrentRequests = 512
```



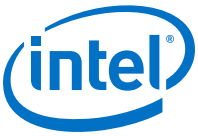
```
Dc0BankNumber = 0
Dc0RingTx = 6
Dc0RingRx = 14

# Data Compression - Kernel instance #1
Dc1Name = "IPComp1"
Dc1IsPolled = 0
Dc1NumConcurrentRequests = 512
Dc1BankNumber = 1
Dc1RingTx = 6
Dc1RingRx = 14


##############################################
# User Process Instance Section
##############################################
[SSL]
NumberCyInstances = 4
NumberDcInstances = 2

# Crypto - User instance #0
Cy0Name = "SSL0"
Cy0BankNumber = 2
Cy0IsPolled = 1
Cy0NumConcurrentSymRequests = 512
Cy0NumConcurrentAsymRequests = 64
Cy0RingAsymTx = 0
Cy0RingAsymRx = 8
Cy0RingSymTx = 2
Cy0RingSymRx = 10
Cy0RingNrbgTx = 4
Cy0RingNrbgRx = 12

# Crypto - User instance #1
Cy1Name = "SSL1"
Cy1BankNumber = 3
Cy1IsPolled = 1
Cy1NumConcurrentSymRequests = 512
Cy1NumConcurrentAsymRequests = 64
Cy1RingAsymTx = 0
Cy1RingAsymRx = 8
Cy1RingSymTx = 2
Cy1RingSymRx = 10
Cy1RingNrbgTx = 4
Cy1RingNrbgRx = 12

# Crypto - User instance #2
Cy2Name = "SSL2"
Cy2BankNumber = 2
Cy2IsPolled = 1
Cy2NumConcurrentSymRequests = 512
Cy2NumConcurrentAsymRequests = 64
Cy2RingAsymTx = 1
Cy2RingAsymRx = 9
Cy2RingSymTx = 3
Cy2RingSymRx = 11
Cy2RingNrbgTx = 5
Cy2RingNrbgRx = 13

# Crypto - User instance #3
Cy3Name = "SSL3"
Cy3BankNumber = 3
Cy3IsPolled = 1
Cy3NumConcurrentSymRequests = 512
Cy3NumConcurrentAsymRequests = 64
Cy3RingAsymTx = 1
Cy3RingAsymRx = 9
Cy3RingSymTx = 3
Cy3RingSymRx = 11
Cy3RingNrbgTx = 5
Cy3RingNrbgRx = 13
```

```
# Data Compression - User instance #0
Dc0Name = "UserDC0"
Dc0BankNumber = 2
Dc0IsPolled = 1
Dc0NumConcurrentRequests = 512
Dc0RingTx = 6
Dc0RingRx = 14

# Data Compression - User instance #1
Dc1Name = "UserDC1"
Dc1BankNumber = 3
Dc1IsPolled = 1
Dc1NumConcurrentRequests = 512
Dc1RingTx = 6
Dc1RingRx = 14
```



Appendix B Glossary

ADF	Acceleration Driver Framework
AHCI	Advanced Host Controller Interface
AP	Application Processor
ASIC	Application Specific Integrated Circuit
Coleto Creek	Codename for the Intel® Communications Chipset 8925 to 8955 Series PCH
Crystal Beach	Codename for a set of chipset functions that allows discrete PCI Express* (PCIe*) adapters to achieve higher performance.
DID	Device ID
DMA	Direct Memory Access
DRGB	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
EHCI	Enhanced Host Controller Interface
GPIO	General Purpose Input Output
GPL	General Public License
IBV	Independent BIOS Vendor
LPC	Low Pincount Interface
MGF	Mask Generation Function
MSI	Message Signaled Interrupts
PCH	Platform Controller Hub. In this manual, a Intel® Communications Chipset 8925 to 8955 Series device that includes standard interfaces and accelerator and I/O interfaces.
RCiEP	Root Complex Integrated Endpoint
RTOS	Real Time Operating System
SAL	Service Access Layer
SATA	Serial Advanced Technology Attachment
SGL	Scatter Gather List
SIO	Serial I/O
SMBus	System Management Bus



SoC	System-on-a-Chip
SPI	Serial Peripheral Interconnect
SR-IOV	Single Root I/O Virtualization
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver/Transmitter
UEFI	Unified Extensible Firmware Interface
UHCI	Universal Host Controller Interface
USB	Universal Serial Bus
WDT	Watch Dog Timer