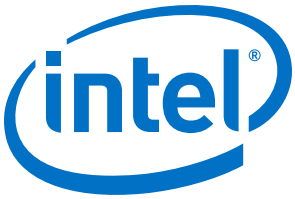


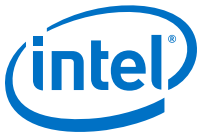
Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure Software

Programmer's Guide

Rev. 002

November 2016

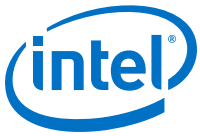
Order No.: 333036-002

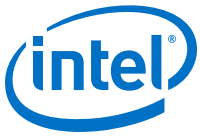


Revision History

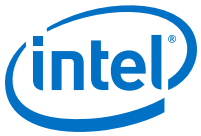
Revision No.	Description of Change	Revision Date
002	Update includes: • Updated Section 1.3 Product Documentation.	November 2016
001	First "public" version of the document. Based on "Intel Confidential" document number 503877-1.3, with the revision history of that document retained for reference purposes Updates include: • Updated Build Flag Summary on page 43 • Updated Intel® QuickAssist Technology API Limitations on page 73	October 2015
1.3	Updates for Revision 1.3: • Added Intel® QuickAssist Technology Entries in the /proc Filesystem on page 36 • Added How to Call the Heartbeat Query on page 38 • Added Acceleration Driver Return Codes on page 47	March 2015
1.2	Updates for Revision 1.2: • Added Utility for Loading Configuration Files and Sending Events to the Driver - adf_ctl on page 28 • Updated Intel® QuickAssist Technology API Limitations on page 73 • Added new APIs to Dynamic Instance Allocation Functions on page 77 and Polling Functions on page 80 • Added Reset Device Function on page 89 • Added Thread-less APIs on page 90 • Other general updates.	January 2015
1.1	Updates for Release 1.1: • Added optional build flag "ICP_NUM_PAGES_PER_ALLOC" to Build Flag Summary on page 43 • Added Running Applications as Non-Root User on page 45	February 2014
1.0	Updates for Release 1.0: • Updated product branding.	September 2013

Contents

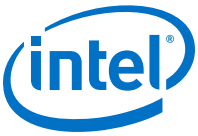




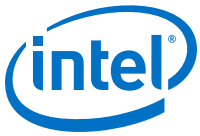
Figures



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 the acceleration service (cryptographic), is documented in the related QuickAssist Technology Software Library documentation (see Product Documentation).

1.1 Terminology

In this document, for convenience:

- Software package is used as a generic term for the Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure software package.
- Accelerator is used as a generic term for the Intel® QuickAssist Accelerator device(s) integrated in the Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure.
- Acceleration drivers is used as a generic term for the software that allows the QuickAssist Software Library APIs to access the Intel® QuickAssist Accelerator device(s) integrated in the Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure.

Refer to Glossary on page 117 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 acceleration driver included in the software package.
- Part 3: 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® Atom™ Processor C2000 Product Family for Communications Infrastructure Software Release Notes*
- *Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure Software for Linux* Getting Started Guide*
- *Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure 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™QuickAssist Technology Acceleration Software OS Porting Guide*
- *Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure External Design Specification (EDS), Vol. 1, 2, 3, and 4*
- Intel® 82580 Quad/Dual Gigabit Ethernet Controller Data Sheet

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

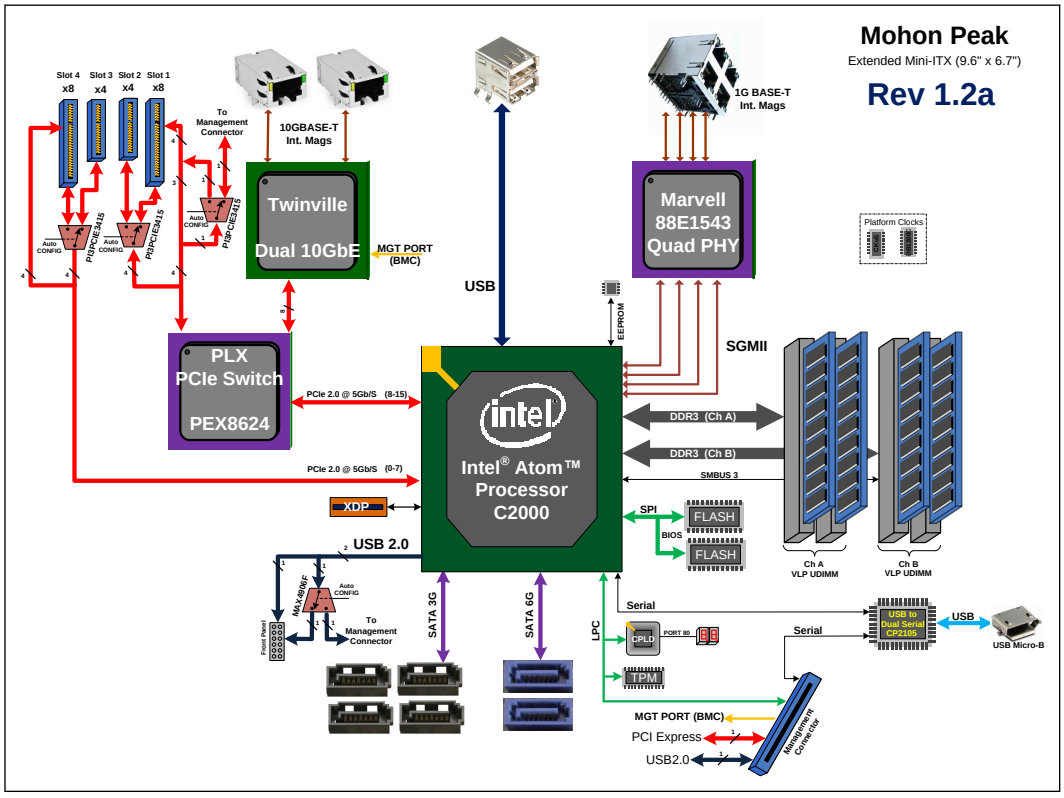
2.0 Platform Overview

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 platform delivers leadership solutions with GB/s Ethernet* MACs and Intel® QuickAssist Technology hardware: the acceleration for cryptography.

2.1 Platform Synopsis

The Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure is a multi-core Intel® Atom™ processor-based SoC offering best Intel architecture (IA) energy efficiency, IO, and Intel QuickAssist Technology integration for embedded/ communications platforms.

Figure 1. Mohon Peak Platform



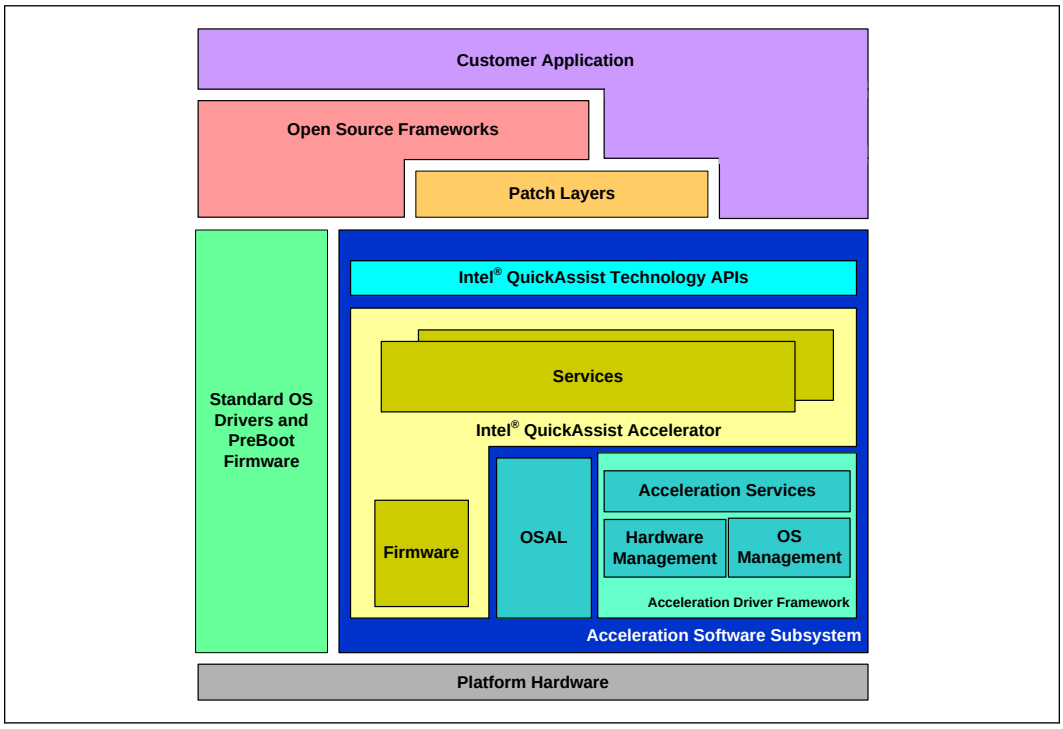
3.0 Software Overview

In addition to the hardware mentioned in Platform Overview on page 13, 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 accelerator in the SoC.

3.1 High-Level Software Architecture Overview

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

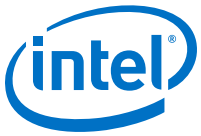
Figure 2. Software Architecture Overview



The main software components are:

- **Pre-boot Firmware**

  The Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure 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. These standard OS drivers are described in Part 2: Acceleration Driver on page 21 of this manual.

- **Acceleration Software Subsystem**

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

  — **Services (Cryptographic)**

  Includes the firmware that drives the various workload slices in the accelerator, and the associated Intel® architecture Service libraries that expose these workloads via APIs. The Service libraries use the Acceleration Driver Framework (ADF) to plug into the OS and gain access to the hardware to communicate with the firmware. The architecture for this subsystem is detailed in : Acceleration Driver on page 21 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 in the SoC. More detail on the APIs and associated architecture is detailed in : "Acceleration Driver" 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 driver. 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 50 for more information.

- **Open Source Frameworks**

  This layer includes open source stacks, such as the Linux Kernel Crypto framework 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 SoC 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 SoC with little or no change provided that the Intel® QuickAssist Technology APIs are integrated with the OS stack or middleware used.

3.2 Logical Instances

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

3.2.1 Response Processing

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

- Interrupt mode
- Polled mode

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

- Polled mode
- Interrupt mode

3.2.1.1 Interrupt Mode

The interrupt is supported in both Kernel and User space.

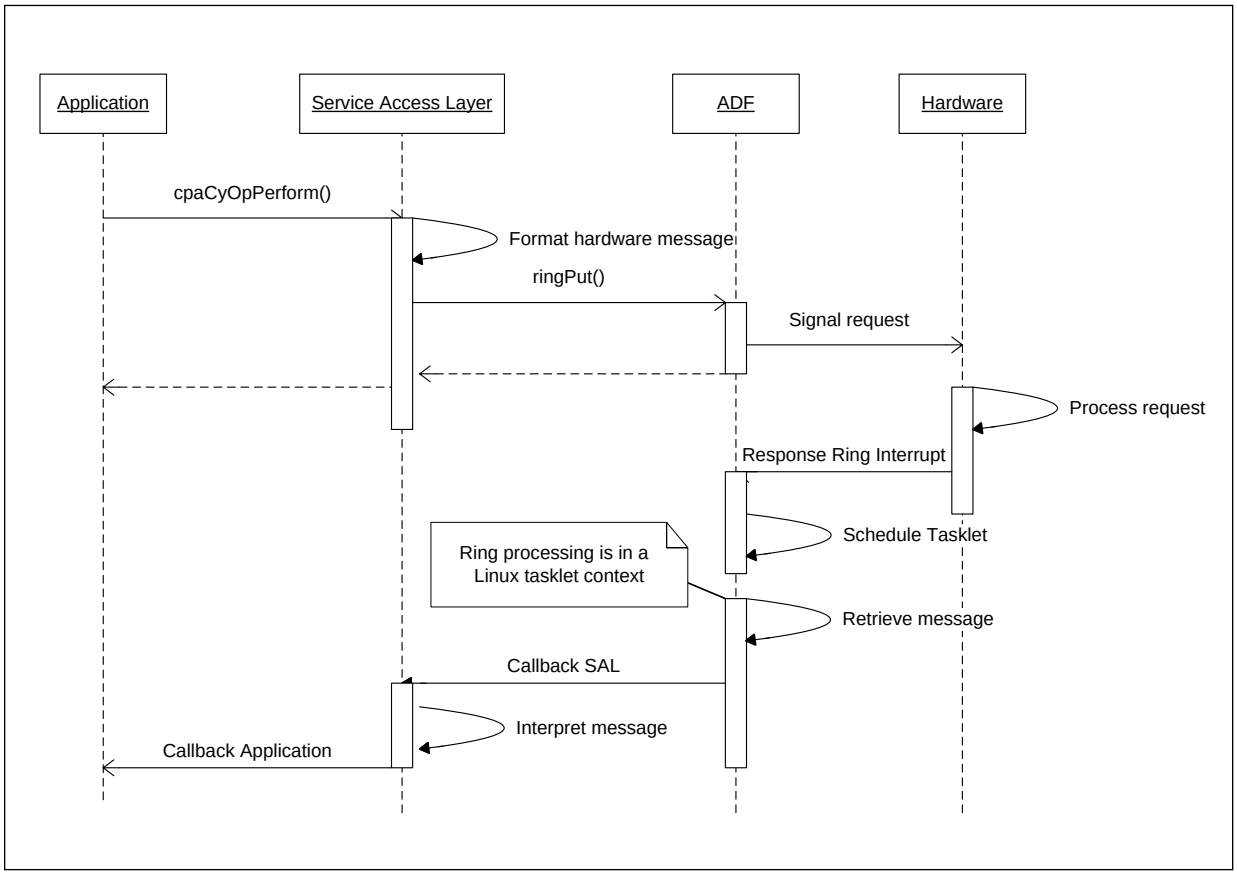
When configured in interrupt mode, the Accelerator Driver Framework (ADF) registers an interrupt handler for response ring processing.

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

The ADF registers an interrupt handler to service the ring bank interrupt. When an interrupt fires, the ADF services the interrupt and creates an interrupt handler bottom 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 Table 4 on page 51 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 3. 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 corresponding or parameter in the configuration file to 1. See Cryptographic Logical Instance Parameters on page 55 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

See Polling Functions for details on all the polling functions.

3.3 Operating System Support

The software package supports the Linux* operating system. Intel® QuickAssist Technology software requires that the following crypto modules be present on the system: `sha256-generic.ko` and `sha512-generic.ko`.

3.4 OpenSSL* Library Inclusion and Usage

The Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure 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® Atom™ Processor C2000 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 28 for additional details.

Software Architecture

The acceleration drivers for Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure 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.

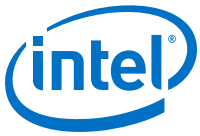
Software Packaging

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® Atom™ Processor C2000 Product Family for Communications Infrastructure 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® Atom™ Processor C2000 Product Family for Communications Infrastructure 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® Atom™ Processor C2000 Product Family for Communications Infrastructure 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® Atom™ Processor C2000 Product Family for Communications Infrastructure devices only.
- If one or more Intel® Communications Chipset 8925 to 8955 Series devices are visible on the PCIe bus and no Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure device is present, the installer.sh will build with support for Intel® Communications Chipset 8925 to 8955 Series devices only.
- If one or more Intel® Communications Chipset 8925 to 8955 Series devices are visible on the PCIe bus and one or more Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure devices are present, the installer.sh will build with support for both Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure devices and Intel® Communications Chipset 8925 to 8955 Series.

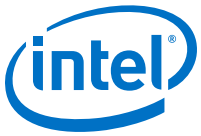
There are two primary usage models for building with support for multiple acceleration hardware generations:

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

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

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

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



Part 2: Acceleration Driver

4.0 Acceleration Driver Overview

For the Cryptographic acceleration service, 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 Acceleration Driver is supported on 64-bit and 32-bit kernels. 32-bit user space applications are supported on 32-bit and 64-bit kernels.

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

4.1 Hardware Assisted Rings

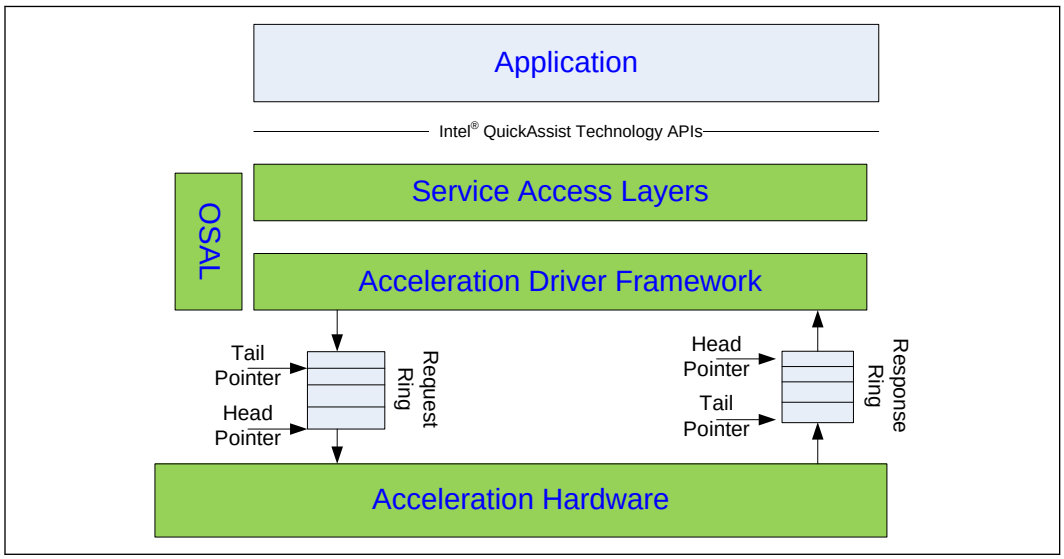
Hardware assisted rings are used as the communication mechanism to transfer requests between the CPU and the accelerator(s) on the chipset device and vice-versa. The hardware supports 128 rings, each with head and tail Configuration Status Register (CSR) pointers that are mapped to PCIe* iRC device (Root Complex integrated device) memory on the CPU. The rings may be configured as:

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

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

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

Figure 4. Intel® QuickAssist Accelerator Ring Access



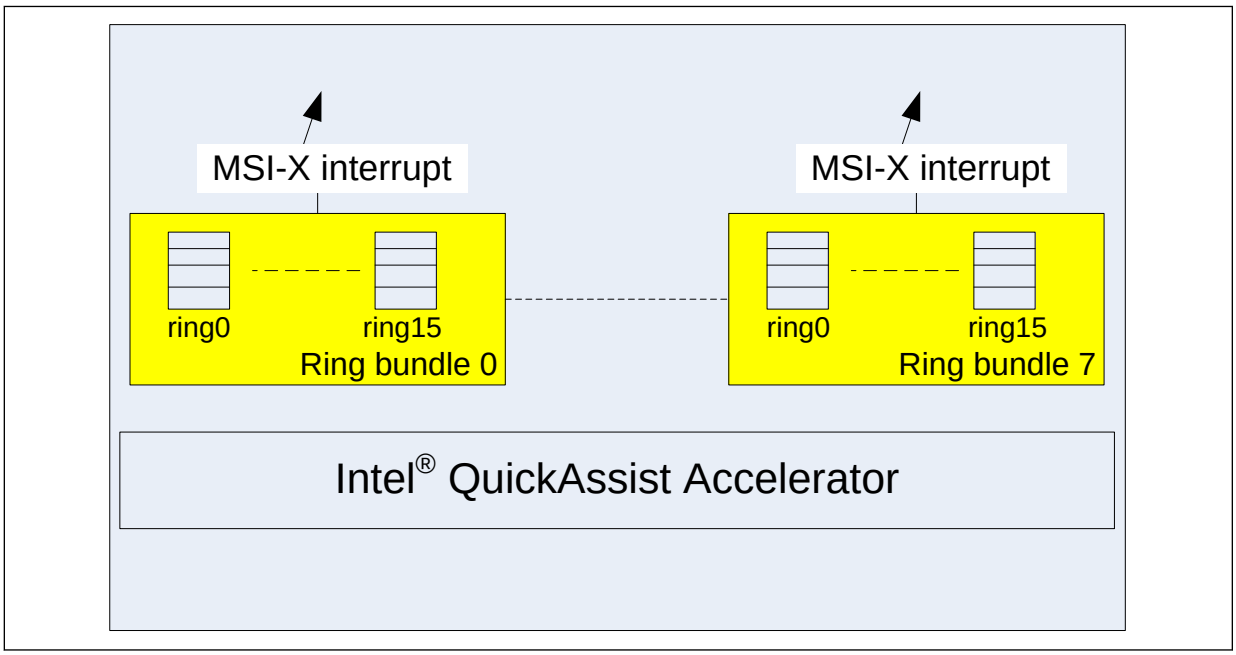
Rings are grouped into *ring banks* with each ring bank containing 16 rings.

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

On the accelerator in the SoC, there are eight independent ring banks. Each ring bank has an associated ring interrupt. If the OS supports MSI-X interrupts, the response may be directed to any core on system. This allows an even distribution of response processing among the cores on the system. The configuration of bank interrupts and core affinity is detailed in Affinity Parameters on page 107.

Depending on the SoC device model number, there is either one or zero accelerators on the device. The following figure shows an overview of the ring banks and accelerator for a single SoC device.

Figure 5. Ring Partitioning on the SoC Device



4.2 Basic Software Context for the Acceleration Driver

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

Figure 6. Basic Software Context

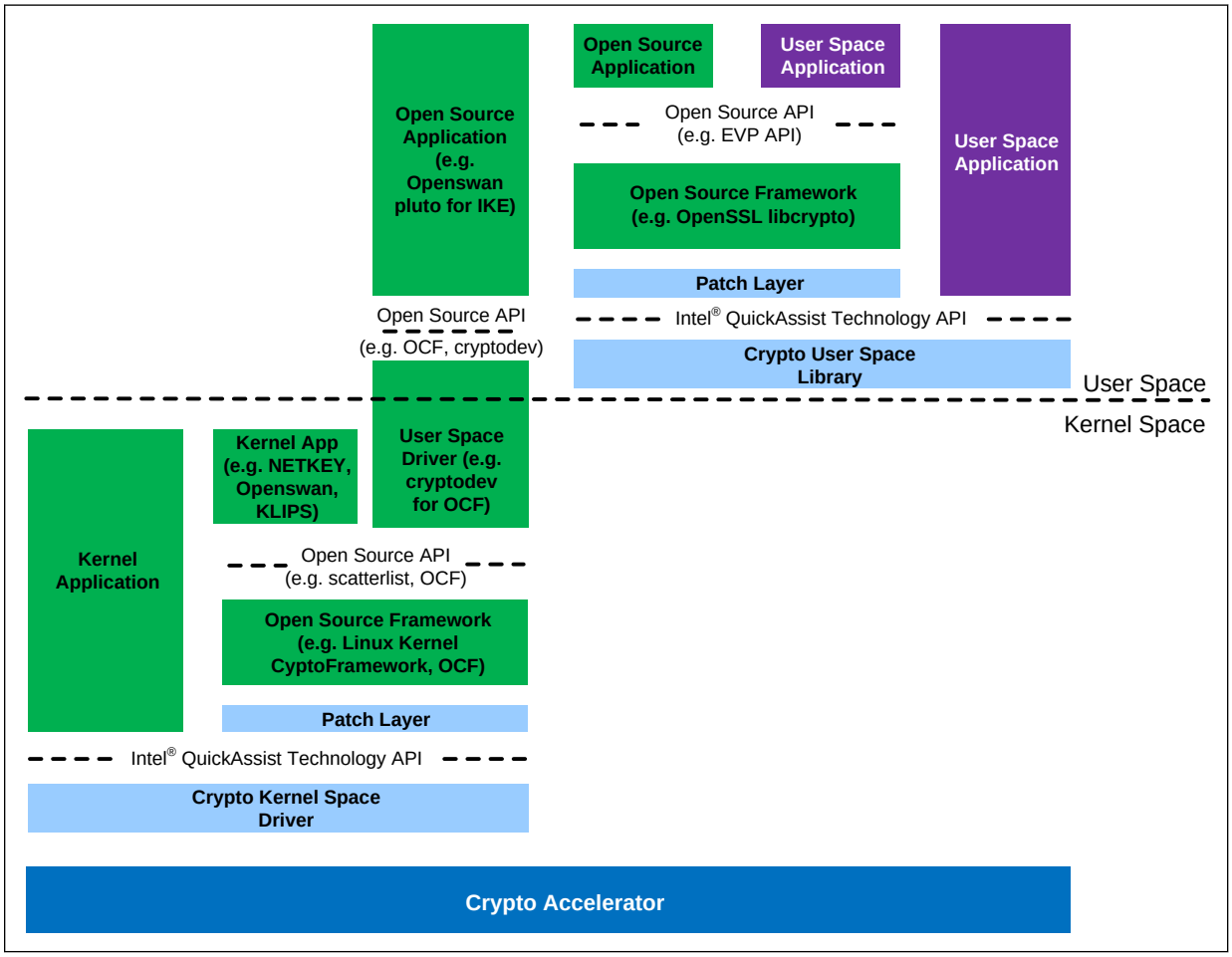
The key elements of this model are as follows:

- The firmware encompasses software executing on the accelerator.
- Intel® architecture software entities that fall into two groups:
  - Driver level entities - CryptoAcc, 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 an accelerator via the Intel® QuickAssist Technology APIs.

4.3 Linux* Software Context for the Acceleration Driver

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 a statement of intent for enabling.

Note: Software packages for patches, such as OpenSSL, Linux Kernel Crypto Framework, NetKey are distributed separately. See Product Documentation on page 11. You will need an Intel Business Link (IBL) account and a subscription to the Electronic Design Kit (EDK).

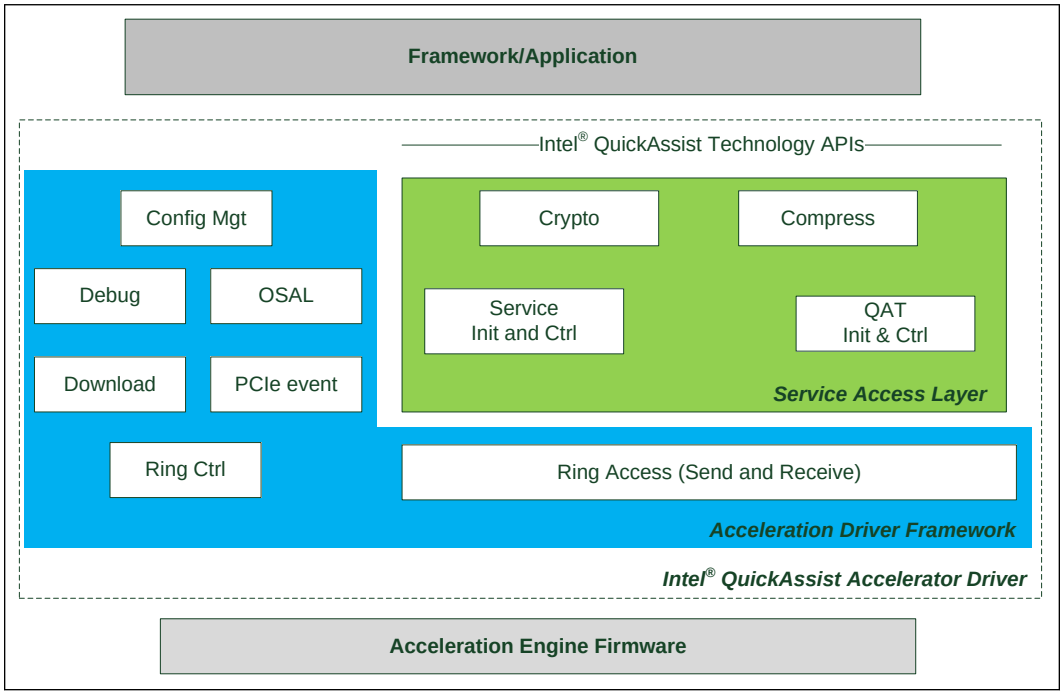
4.4 NUMA Support

The Acceleration Driver software is implemented with the Non-uniform Memory Access (NUMA) architecture in mind. For optimal performance, it is important to ensure that any memory passed to the acceleration device via the driver APIs is allocated from the NUMA node that is local to the acceleration device.

4.5 Acceleration Driver

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



Note: Compression is not supported by the Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure.

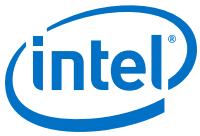
4.5.1 Framework Overview

An acceleration driver contains a number of logical units that are primarily exposed via the Intel® QuickAssist Technology APIs. Figure 8 on page 26 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)**



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

4.5.2 Service Access Layer

The Service Access Layer (SAL) is responsible for providing access to the individual crytographic service contained in the accelerator. As shown in Figure 8 on page 26, the layer is made up of the individual services as well as an Initialization and Control component.

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

The primary responsibilities of this layer are as follows:

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

4.5.3 Acceleration Driver Framework

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

Services include:

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

- **OS Abstraction**: The SAL layer is OS independent and makes use of the OSAL provided as part of the ADF.

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

4.5.4 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 SoC device in the system. The configuration file format is described in Acceleration Driver Configuration File on page 50. The older legacy configuration file format (which is still supported) is described in Acceleration Driver Configuration File - Earlier File Format on page 102.

4.5.5 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

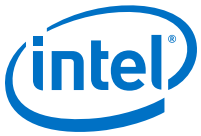
Device Enumeration

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

continued...



Driver	adf_ctl status			Conf File Name	API	
						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 89xx Series Software for Linux* Getting Started Guide*.

Case where only DH895xcc devices are on the platform

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

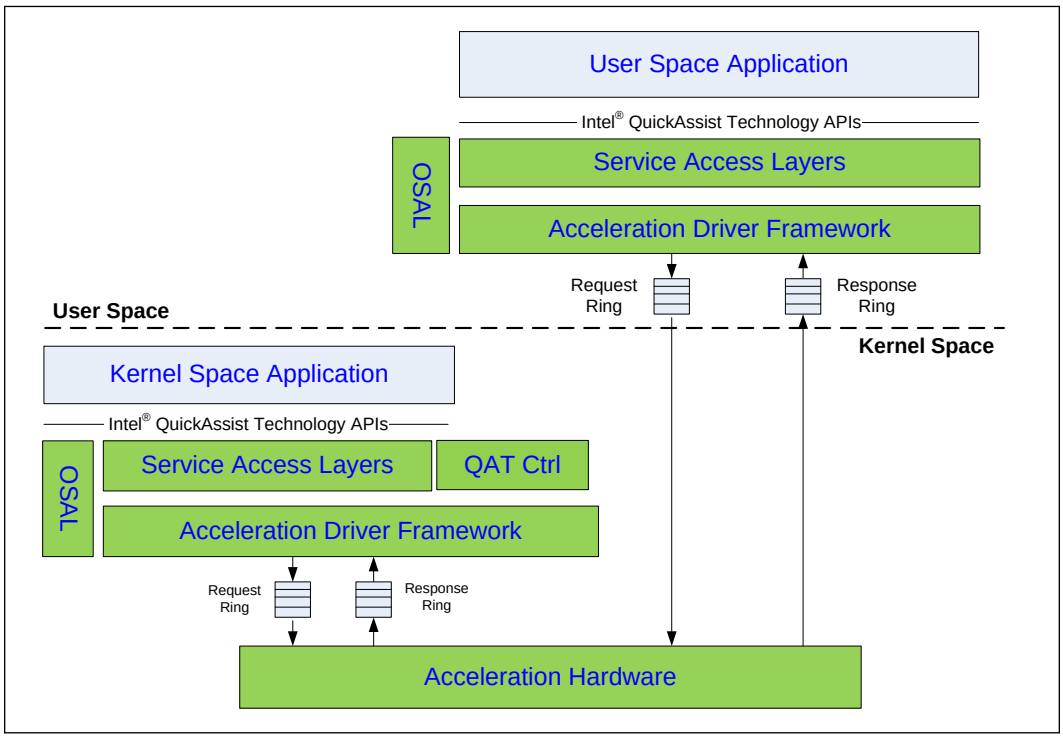
Case where DH895xcc and DH89xxcc devices are on the platform

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

4.6 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.

4.6.1 User Space Memory Allocation

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

- Accelerator driver memory allocation
- Application payload memory allocation

4.6.1.1 Accelerator Driver Memory Allocation

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

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

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

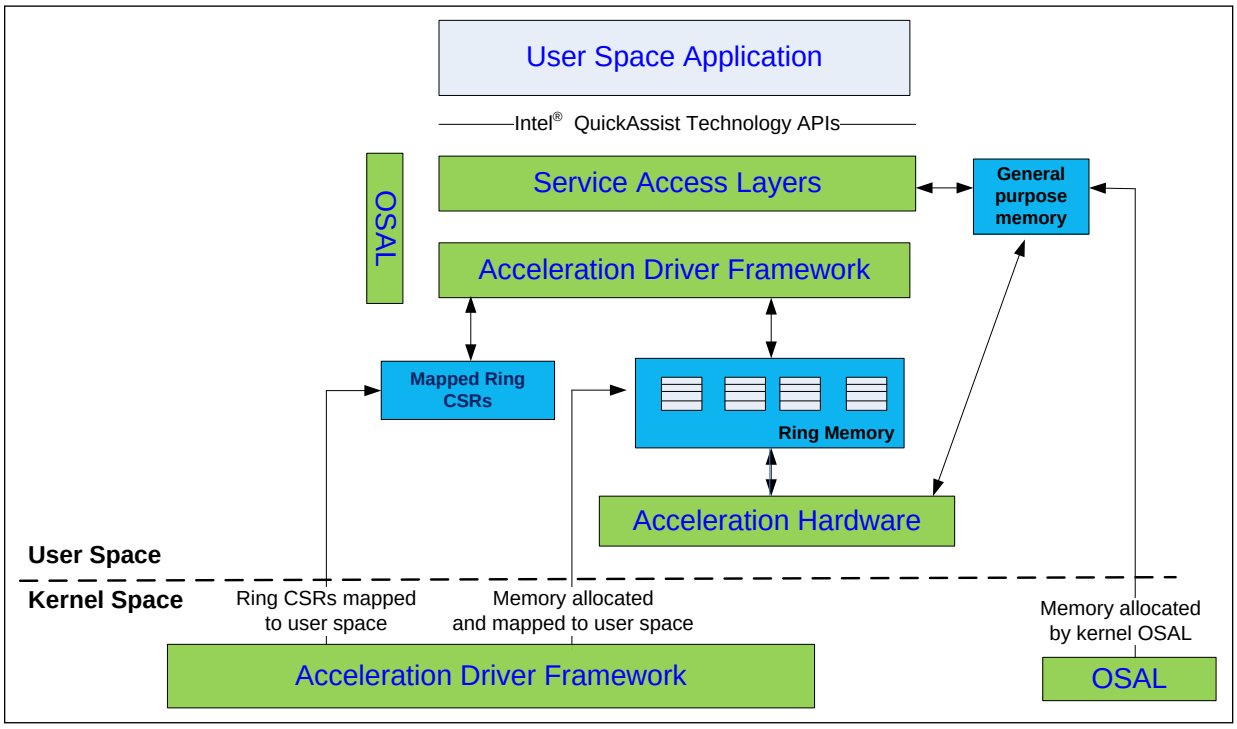
`icp_dev_mem` that may be called through an `IOCTL` function by the OSAL in user space to allocate memory. When called with `IOCTL DEV_MEM_IOC_MEMALLOC`, the OSAL kernel mode driver returns the allocated memory.

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

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

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

Figure 10. User Space Memory Allocation at Initialization



4.6.1.2 Application Payload Memory Allocation

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

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

Cryptographic service	`cpaCySetAddressTranslation`	See the *Intel® QuickAssist Technology Cryptographic 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.

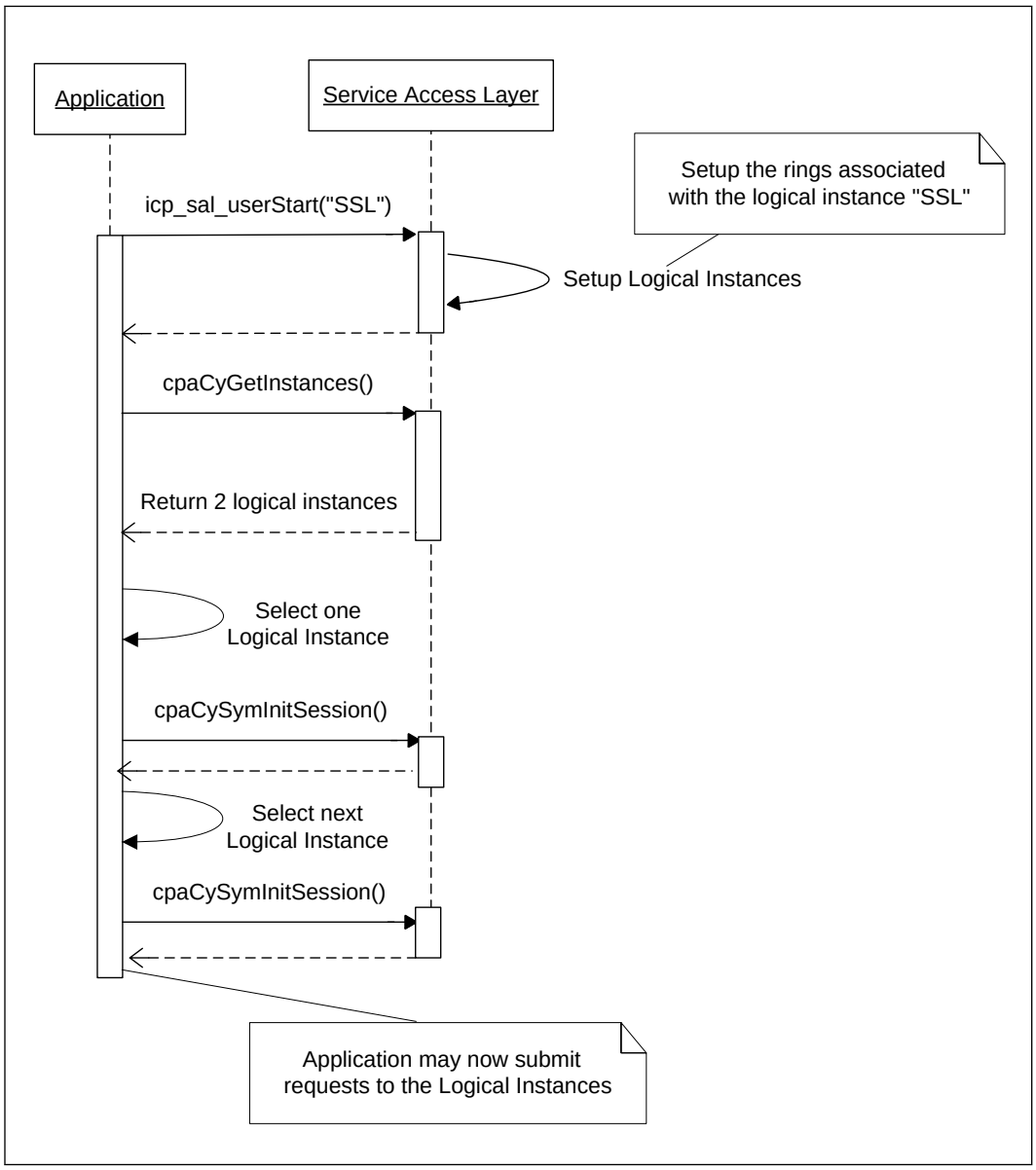
4.6.2 User Space Additional Functions

To allow a user space process access to the Intel® QuickAssist Accelerator rings, the service access layer needs to be configured to expose logical instances to the user space process. Logical instances are configured using the per device configuration file. See User Space Configuration on page 33 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 85 function (or icp_sal_userStart on page 85 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 87.

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



4.6.3 User Space Configuration

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

For example, in the sequence in Figure 11 on page 33, 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]
############################################
# User Process Instance Section
############################################
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
```

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

4.6.4 User Space Response Processing

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

- Polled mode
- Interrupt mode

4.6.4.1 User Space Interrupt Mode

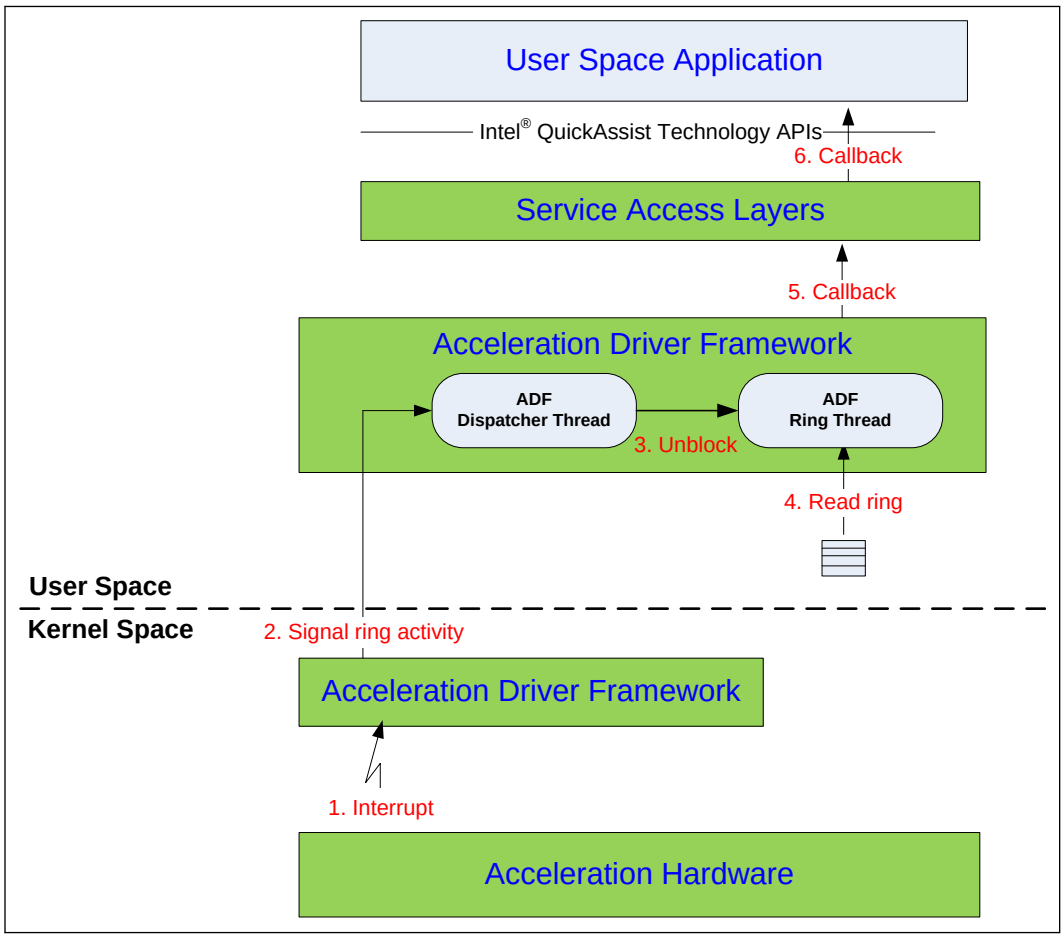
Note: User space interrupt mode is being removed from future Intel® QuickAssist Technology releases. A new event-based user space notification mechanism will be added. Please discuss any concerns with your Intel representative.

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

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

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

Figure 12. User Space Response Processing for Interrupt Mode



4.6.4.2 User Space Polled Mode

The sequence for user space polling does not differ from that described in Polled Mode on page 17.

4.7 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
```

The output will be similar to the following:

```
icp_dev0 is up
```

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

4.8 Intel® QuickAssist Technology Entries in the /proc Filesystem

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

/proc/icp_c2xxx_devX/ files, where X is the device number	Description of information contained in that file
./cfg_debug	Internal configuration table generated from: /etc/c2xxx_qa_devX.conf and from some internal data, e.g., firmware version. It is useful to check which user processes and instances have been configured.

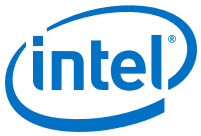
.qatX	Statistics for Intel® QuickAssist Technology (QAT), overall number of requests/ responses per ME. FW is loaded on each ME, if ME 0 gets one request, processes it and put it back on the ring, then the FW counters for Request and Response will be incremented by 1 for that ME. Example output for one ME is:
	`+----------------------------------------------------+`
	`| Statistics for Qat Instance 0 |`
	`+----------------------------------------------------+`
	`| Firmware Requests[AE 0]: 1 |`
	`| Firmware Responses[AE 0]: 1 |`
	For QAT 1.5 and QAT 1.6, this also triggers the heartbeat query below.

./version	Lists hardware, software and API versions in use. Example output for QAT1.6:
	`+----------------------------------------------------+`
	`| Hardware and Software versions for device 0 |`
	`+----------------------------------------------------+`
	`Hardware Version: B0 SKU2`
	`Firmware Version: 1.4.0`
	`MMP Version: 1.0.0`
	`Driver Version: 1.8.1`
	`QuickAssist API CY Version: 1.8`
	`QuickAssist API DC Version: 1.4`
	`+----------------------------------------------------+`

./cy/IPSecY	For cy stats, see Section 5.2.2

./et_ring_ctrl/bank_Y/conf	Refers to EagleTail_Ring_Control, this conf file gives a summary on all EagleTailRings in use in bank_Y, where Y is one of the banks configured for use.
	Example output:
	`cat`
	`/proc/icp_dh895xcc_dev0/et_ring_ctrl/bank_0/conf`
	`------- Bank 0 Configuration -------`
	`Interrupt Coalescing Enabled`
	`Interrupt Coalescing Counter = 10000`
	`Interrupt mask: 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0`
	`User interrupt mask: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0`
	`Polling mask: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0`

continued...



/proc/icp_c2xxx_devX/ files, where X is the device number	Description of information contained in that file
	Coalesc reg: 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 Bank empty stat: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 Bank nempty stat: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ------- Rings: Ring Number: 0, Config: 80000006, Base Addr: ffff880267e50000 Head: 0, Tail: 0, Space: 1000, inflights: 0, Name: Cy0RingAsymTx Ring Number: 2, Config: 8000000a, Base Addr: ffff88021ea60000 Head: 0, Tail: 0, Space: 10000, inflights: 0, Name: Cy0RingSymTx Ring Number: 4, Config: 8000000a, Base Addr: ffff88021e8a0000 Head: 0, Tail: 0, Space: 10000, inflights: 0, Name: Cy0RingNrbgTx Ring Number: 6, Config: 8000000a, Base Addr: ffff88021ffd0000 Head: 0, Tail: 0, Space: 10000, inflights: 0, Name: Dc0RingTx Ring Number: 8, Config: 5405, Base Addr: ffff880267e51000 Head: 0, Tail: 0, Space: 1000, inflights: 0, Name: Cy0RingAsymRx Ring Number: 10, Config: 5408, Base Addr: ffff880220140000 Head: 0, Tail: 0, Space: 4000, inflights: 0, Name: Cy0RingSymRx Ring Number: 12, Config: 5408, Base Addr: ffff8802200cc000 Head: 0, Tail: 0, Space: 4000, inflights: 0, Name: Cy0RingNrbgRx Ring Number: 14, Config: 5408, Base Addr: ffff8802202b4000 Head: 0, Tail: 0, Space: 4000, inflights: 0, Name: Dc0RingRx ------------------------------------
./et_ring_ctrl/bank_Y/ ring_Z	Gives information on each specific ring. For example ring_0 from the above conf entry will give the data on that ring and accelerator number associated with it in addition to the information given in the conf entry: ------- Ring Configuration ------- Service Name: Cy0RingAsymTx Accelerator Number: 0, Bank Number: 0, Ring Number: 0 Ring Config: 80000006 Tx, Base Address: ffff880267e50000, Head: 0, Tail: 0, Space: 1000 Message size: 64, Max messages: 64, Current messages: 0 Ring Empty flag: 1, Ring Nearly Empty flag: 1 ----------- Ring Data ----------- Memory Address: <Contents of memory address (64bytes)>

4.9 Heartbeat Feature and Recovery from Hardware Errors

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

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

Note: Recovery on detection of a Heartbeat failure is not enabled by default. Automatic recovery can be enabled by building the acceleration software with a compile-time flag.

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

4.9.1 How to Call the Heartbeat Query

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

- Periodically call heartbeat APIs (see User Application Heartbeat APIs (not Enabled by Default) on page 39).

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

4.9.2 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:

- Manually from command line using the command:

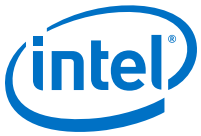
```
# cat /proc/icp_sxxxx_dev0/qat0
```

- From a watch process running in background:

```
# watch -n0.1 cat /proc/icp_sxxxx_dev0/qat0 > /dev/null
```

- From simple script running in the background:

```
#!/bin/bash
while :
do
  cat  /proc/icp_sxxxx_dev0/qat0 > /dev/null
  sleep 1
done
```



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

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

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

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

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

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

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

4.9.3 User Application Heartbeat APIs (not Enabled by Default)

These functions have the following signatures:

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

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

See icp_sal_check_device on page 88 and icp_sal_check_all_devices on page 88 for details on the functions and parameters.

4.10 Driver Threading Model

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

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

- **Monitor Thread**

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

- **Listener Thread**

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

- **Ring Thread**

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

4.10.1 Thread-less Mode

The user sets an environment variable:

```
setenv ICP_WITHOUT_THREAD = 1
```

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

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

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

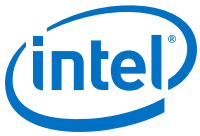
- `CpaStatus icp_sal_find_new_devices(void)` - Performs a function similar to the monitor thread, that is, checks if there are new devices in the system.
- `CpaStatus icp_sal_poll_device_events(void)` - Performs a function similar to the listener thread, that is, polls for events.

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

4.11 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.

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

For kernel space instances, the same information can be obtained from the `/proc` file system if the required statistics parameters are enabled in the configuration file, as the following configuration file extract shows. See also Statistics Parameters on page 53 for more detail.

```
#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
```

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_c2xxx_dev0/cy

total 0

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

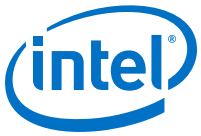
The statistics can then be queried simply by running `cat` on the corresponding file in the `/proc` file system. For example:

```
# cat /proc/icp_c2xxx_dev0/cy/IPSec0
```

The output is similar to the following:

```
+-------------------------------------------------+
| Statistics for Instance                  IPSec0 |
| Symmetric Stats                                 |
+-------------------------------------------------+
```

```
| Sessions Initialized:                          86 |
| Sessions Removed:                              86 |
| Session Errors:                                 0 |
+------------------------------------------------+
| Symmetric Requests:                           960 |
| Symmetric Request Errors:                       0 |
| Symmetric Completed:                          960 |
| Symmetric Completed Errors:                     0 |
| Symmetric Verify Failures:                      0 |
+------------------------------------------------+
| DSA Stats                                      |
+------------------------------------------------+
| DSA P Param Gen Requests-Succ:                  0 |
| DSA P Param Gen Requests-Err:                   0 |
| DSA P Param Gen Completed-Succ:                 0 |
| DSA P Param Gen Completed-Err:                  0 |
+------------------------------------------------+
| DSA G Param Gen Requests-Succ:                  1 |
| DSA G Param Gen Requests-Err:                   0 |
| DSA G Param Gen Completed-Succ:                 1 |
| DSA G Param Gen Completed-Err:                  0 |
+------------------------------------------------+
| DSA Y Param Gen Requests-Succ:                 20 |
| DSA Y Param Gen Requests-Err:                   0 |
| DSA Y Param Gen Completed-Succ:                20 |
| DSA Y Param Gen Completed-Err:                  0 |
+------------------------------------------------+
| DSA R Sign Requests-Succ:                       0 |
| DSA R Sign Request-Err:                         0 |
| DSA R Sign Completed-Succ:                      0 |
| DSA R Sign Completed-Err:                       0 |
+------------------------------------------------+
| DSA S Sign Requests-Succ:                       0 |
| DSA S Sign Request-Err:                         0 |
| DSA S Sign Completed-Succ:                      0 |
| DSA S Sign Completed-Err:                       0 |
+------------------------------------------------+
| DSA RS Sign Requests-Succ:                     20 |
| DSA RS Sign Request-Err:                        0 |
| DSA RS Sign Completed-Succ:                    20 |
| DSA RS Sign Completed-Err:                      0 |
+------------------------------------------------+
| DSA Verify Requests-Succ:                      20 |
| DSA Verify Request-Err:                         0 |
| DSA Verify Completed-Succ:                     20 |
| DSA Verify Completed-Err:                       0 |
| DSA Verify Completed-Failure:                   0 |
+------------------------------------------------+
| RSA Stats                                      |
+------------------------------------------------+
| RSA Key Gen Requests:                          20 |
| RSA Key Gen Request Errors                      0 |
| RSA Key Gen Completed:                         20 |
| RSA Key Gen Completed Errors:                   0 |
+------------------------------------------------+
| RSA Encrypt Requests:                           0 |
| RSA Encrypt Request Errors:                     0 |
| RSA Encrypt Completed:                          0 |
| RSA Encrypt Completed Errors:                   0 |
+------------------------------------------------+
| RSA Decrypt Requests:                          20 |
| RSA Decrypt Request Errors:                     0 |
| RSA Decrypt Completed:                         20 |
| RSA Decrypt Completed Errors:                   0 |
+------------------------------------------------+
| Diffie Hellman Stats                           |
+------------------------------------------------+
| DH Phase1 Key Gen Requests:                    40 |
| DH Phase1 Key Gen Request Err:                  0 |
| DH Phase1 Key Gen Completed:                   40 |
| DH Phase1 Key Gen Completed Err:                0 |
```



```
+-------------------------------------------------+
| DH Phase2 Key Gen Requests:                  40 |
| DH Phase2 Key Gen Request Err:                0 |
| DH Phase2 Key Gen Completed:                 40 |
| DH Phase2 Key Gen Completed Err:              0 |
+-------------------------------------------------+
| Key Stats                                       |
+-------------------------------------------------+
| SSL Key Requests:                             0 |
| SSL Key Request Errors:                       0 |
| SSL Key Completed                             0 |
| SSL Key Complete Errors:                      0 |
+-------------------------------------------------+
| TLS Key Requests:                             0 |
| TLS Key Request Errors:                       0 |
| TLS Key Completed                             0 |
| TLS Key Complete Errors:                      0 |
+-------------------------------------------
```

4.12 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

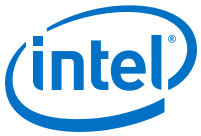
Symbol	Description	Default	Reference
ICP_ROOT	Set to the directory where acceleration software is extracted. This may be /QAT or /QAT/QAT1.5, depending on how the driver was compiled.	User defined	
ICP_BUILDSYSTEM_PATH	Set to the build system folder located under the quickassist folder ($ICP_ROOT/quickassist/build_system)	User defined	
ICP_BUILD_OUTPUT	Set to directory that executable/ libraries are placed in ($ICP_ROOT/build)	User defined	
ICP_ENV_DIR	Set to the directory that contains the environmental build files ($ICP_ROOT/quickassist/build_system/build_files/env_files)	User defined	
ICP_TOOLS_TARGET	Set to accelcomp for Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure platforms	User defined	
ICP_A0_C2XXX_SILICON=1	This needs to be defined when compiling for SoC A0 silicon. It is not defined by default. This is set properly if the installation script is used to install the software.	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.	
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 40
ICP_HEARTBEAT	When defined, enables automatic device reset on failures detected by the heartbeat mechanism.		Heartbeat Feature and Recovery from Hardware Errors on page 37
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. *Note:* Not supported for the Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure	Not defined	
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 the console for user space applications OR to /var/log/ messages in kernel space.	Not defined	
LAC_HW_PRECOMPUTES	When defined, enables hardware for HMAC precomputes.	Not defined, therefore the driver uses software (dependency on OpenSSL and Linux Crypto API.	See limitations below table.
max_mr	Used to set the number of Miller Rabin rounds for prime operations. Setting this to a smaller value reduces the memory usage required by the driver.	50	
WITH_CPA_MUX	When defined, the driver will be built for the mux environment, i.e., cpa_mux.ko will be built and will expose the Intel® QuickAssist	Depends on devices found on the platform. Not defined if devices	

continued...



Symbol	Description	Default	Reference
	Technology API. The drivers will not export symbols but will instead register with the cpa_mux.	found can be supported by a single driver. Defined otherwise, e.g., if both DH89xxcc and DH895xcc devices are found.	
ICP_DISABLE_INLINE	When defined, function inlining for functions that cannot be inlined by the compiler is removed to enable compilation of the driver for kernels build without CONFIG_ARCH_SUPPORTS_OPTIMIZED_INLINING	Not defined	

Notes: The limitations of pre-computes are as follows:

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

4.13 Running Applications as Non-Root User

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

Assumptions:

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

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

1. Export environmental variables.

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

   Note: This step is not applicable to Yocto.

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

```
# groupadd <group_name>
```

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

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

4. Change group ownership of the following files. By default, the group ownership will be root.
   - /dev/icp_dev_processes
   - /dev/icp_dev<N>_ring
   - /dev/icp_dev<N>_csr
   - /dev/icp_adf_ctl
   - /dev/icp_dev_mem
   - /dev/icp_dev_mem_page

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

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

```
# chmod –R 644 $ICP_ROOT
# chmod 644 /etc/c2xxx_qa_dev?.conf
```

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

   For 64-bit OS:

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

   For 32-bit OS:

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

   For Yocto:

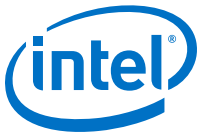
```
# cd /usr/lib
# chgrp <group_name> libicp_qa_al_s.so
```

7. Change the group ownership for memory driver.

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

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

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



9. Launch the performance sample code.

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

For Yocto:

```
# /usr/bin/cpa_sample_code signOfLife=1
```

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

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

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

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

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

4.14 Acceleration Driver Return Codes

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

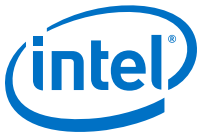
Return Type	Return Code	Description
CPA_STATUS_SUCCESS	0	Requested operation was successful.
CPA_STATUS_FAIL	-1	General or unspecified error occurred. Refer to the console log user space application or to /var/log/messages in kernel space for more details of the failure.
CPA_STATUS_RETRY	-2	Recoverable error occurred. Refer to relevant sections of the API for specifics on what the suggested course of action.
CPA_STATUS_RESOURCE	-3	Required resource is unavailable. The resource that has been requested is unavailable. Refer to relevant sections of the API for specifics on what the suggested course of action.
CPA_STATUS_INVALID_PARAM	-4	Invalid parameter has been passed in.

continued...

Return Type	Return Code	Description
CPA_STATUS_FATAL	-5	Fatal error has occurred. A serious error has occurred. Recommended course of action is to shutdown and restart the component.
CPA_STATUS_UNSUPPORTED	-6	The function is not supported, at least not with the specific parameters supplied. This may be because a particular capability is not supported by the current implementation.
CPA_STATUS_RESTARTING	-7	The API implementation is restarting. This may be reported if, for example, a hardware implementation is undergoing a reset.

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

Return Type	Return Code	Description
SUCCESS	0	Operation was successful.
FAIL	1	General error occurred. Refer to the console log user space application or to /var/log/ messages in kernel space for more details of the failure.
-EPERM	-1	Operation is not permitted. Used during ioctl operations.
-EIO	-5	Input/Output error occurred. Used when copying configuration data to and from user space.
-EBADF	-9	Bad File Number. Used when an invalid file descriptor is detected.
-EAGAIN	-11	Try Again. Used when a recoverable operation occurred.
-ENOMEM	-12	Out of Memory. Memory resource that has been requested is not available.
-EACCES	-13	Permission Denied. Used when the operation failed to connect to a process or open a device.
-EFAULT	-14	Bad Address. Used when an operation detects invalid parameter data.
-ENODEV	-19	No Such Device. Used when an operation detects invalid device id.
-ENOTTY	-25	Invalid Command Type. Used when an ioctl operation detects an invalid command type.



4.15 Compiling Acceleration Software on Older Kernels

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

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

- **Installing**

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

  — `LAC_HW_PRECOMPUTES=1`

  — `ICP_NUM_PAGES_PER_ALLOC=1`

  Once these are defined, compile and install the driver.

- **Testing**

  Once the driver is installed, performance sample code signOfLife tests can be executed. Please refer to the *Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure Software for Linux* Getting Started Guide* for details.

- **Limitations**

  — Older kernels do not support kmalloc of more than 128K. Due to this limitation, compression tests within the performance sample code may not execute.

  — Running the performance sample code without the `signOfLife=1` option may fail.

  — Ensure `LAC_HW_PRECOMPUTES` is defined if your application uses algorithm chaining from kernel space. The acceleration driver by default makes use of software based hashing for algorithm chaining and this functionality was not available in older kernels. Setting the `LAC_HW_PRECOMPUTES` allows the driver to use hardware acceleration.

5.0 Acceleration Driver Configuration File

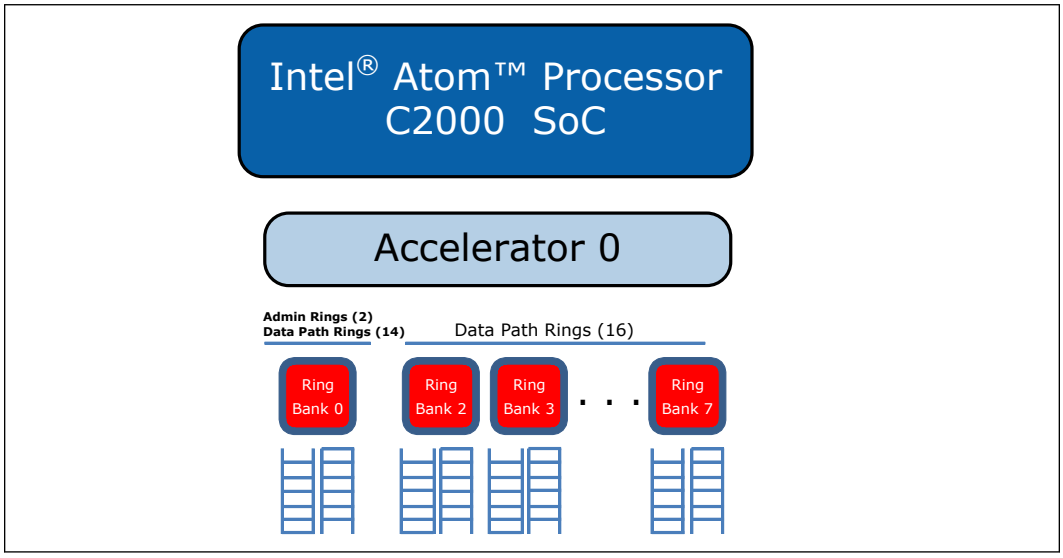
This chapter describes the configuration file(s) managed by the Acceleration Driver Framework (ADF) that allow customization of runtime operation. This configuration file(s) must be tuned to meet the performance needs of the target application.

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

5.1 Configuration File Overview

There is a single configuration file for the Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure device. The accelerator has eight independent ring banks - the communication mechanism between the Acceleration software and the hardware. Each ring bank has an interrupt that can be directed to a specific Intel® architecture core. Each ring bank has 16 rings (hardware assisted queues). This hierarchy is shown in the following figure.

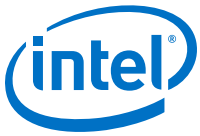
Figure 13. Ring Banks



Note: Depending on the model number, the SoC 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 51.

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

- **Logical Instances** - one or more sections that include parameters that allow the user to set:
  - The number of cryptography 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 54.

A sample configuration file, targeted at a high-end IPsec box, is included in Sample Configuration File (V2) on page 59.

5.2 General Section

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

5.2.1 General Parameters

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

Table 4. General Parameters

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

continued...

Parameter	Description	Default	Range
			Note: X can be 0 or 1, which identifies one of two available cryptographic engines (depending on the SKU) . *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	Name of the Microcode Object File (MOF) firmware.	mof_firmware_c2xxx.bin	mof_firmware_c2xxx.bin
Firmware_MmpPath	Name of the Modular Math Processor (MMP) firmware.	mmp_firmware_c2xxx.bin	mmp_firmware_c2xxx.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
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
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.			

5.2.2 Statistics Parameters

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

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

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

- Calling the appropriate Intel® QuickAssist Technology API function. For example, `cpaCySymQueryStats` or `cpaCySymQueryStats64` for symmetric cryptography. See the *Intel® QuickAssist Technology Cryptographic API Reference Manual* for more information about these functions.
- For kernel space instances, looking at entries in the `/proc/icp_c2xxx_devX` directory, where X is the device number. For example, `/proc/icp_c2xxx_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 40 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
			continued...

Parameter	Description	Default	Range
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".			

5.2.3 Optimized Firmware for Wireless Applications

When using the simplified configuration file format (indicated by the existence of the `ConfigVersion` parameter), if the `NumProcesses` parameter in the [WIRELESS] section of the configuration file is greater than 0, a version of the firmware optimized for small cryptography packets is automatically selected. In this case, each cryptography process consumes six rings as in the "standard" firmware case. The range for the `NumProcesses` parameter in the [WIRELESS] section is constrained in the same way as that describe in Maximum Number of Process Calculations on page 58).

The optimized firmware operates with the following constraints and characteristics:

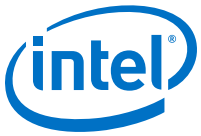
- SGL and Flat buffers are supported.
- The maximum supported Source/Destination payload size is 2K (where payload is either a flat buffer with a size up to 2K or the total number of bytes in flat buffers specified in SGL descriptors.
- Only rings 0-31 are used, i.e., first two banks *where each bank has 16 rings*.
- There is no support for the runtime (resent) `Init AE` and `Init Ring` info messages (these messages must be sent once in the start-up phase per AE).
- Cipher Only and Auth Only (Mode0/Mode1/Mode2) processing is supported.
- 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.
- Request ordering is always enabled.

5.3 Logical Instances Section

This section allows the configuration of logical instances in each address domain (kernel space and individual user space processes). See Hardware Assisted Rings on page 22 and Logical Instances on page 16 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 84 for more information.

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

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

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

5.3.1 [KERNEL] Section

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

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

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

Parameter	Description	Default	Range
NumberCyInstances	Specifies the number of cryptographic instances. *Note:* Depends on the number of allocations to other services.	2	0 to 32
Note: "Default" denotes the value in the configuration file when shipped.			

5.3.1.1 Cryptographic Logical Instance Parameters

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

Table 6. Cryptographic Logical Instance Parameters

Parameter	Description	Default	Range
CyXName	Specifies the name of cryptographic instance number X.	IPSec0	String (max. 64 characters)
CyXAcceleratorNumber	Specifies the accelerator number that the cryptographic instance number X is assigned to.	0	0, 1, 2 or 3
CyXIsPolled	Specifies if cryptographic instance number X works in poll mode or IRQ mode.	0 for kernel space instances	For instance in the kernel space: 0 for IRQ

continued...

Parameter	Description	Default	Range
		1 for user space instances	1 for poll mode For instance in the user space: 0 for IRQ 1 for poll mode
CyXNumConcurrentSymRequests (optional)	Specifies the number of in-progress cryptographic concurrent symmetric requests (and responses) for cryptographic instance number X. *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. *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.	N/A	64, 128, 256, 512, 1024, 2048 or 4096
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.			

5.3.2 [DYN] Section

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

The parameters that can be included in the [DYN] section are the same as those that can be included in the [KERNEL] section. See [KERNEL] Section on page 55 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 77 for more information.

5.3.2.1 Dynamic Instance Configuration Example

The following configuration file snippets demonstrate the reservation of instances for dynamic allocation. In a system that uses the two configuration files below, `icp_sal_userCyInstancesAlloc` can allocate up to 26 cryptographic (cy) instances. See Dynamic Instance Allocation Functions on page 77 for more information.

Taken from: `/etc/c2xxx_qa_dev0.conf`

```
...

[DYN]
NumberCyInstances = 10


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

# Crypto - User instance DYN #1
Cy1Name = "DYN1"
Cy1IsPolled = 1
Cy1AcceleratorNumber = 1
# List of core affinities
Cy1CoreAffinity = 1

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

Taken from: `/etc/c2xxx_qa_dev1.conf`

```
...

[DYN]
NumberCyInstances = 16


...
```

5.3.3 User Process [xxxxx] Sections

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

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

Table 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 86 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 58.
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.		0 to 32
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 54.

5.3.3.1 Maximum Number of Process Calculations

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

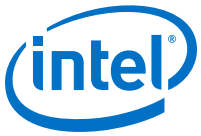
The value to which this parameter can be set is determined by a number of factors, most significantly, the number of cryptography instances in the process section. The total number of processes, per service, created by the driver is given by the expression (e.g., for cryptography):

```
+ NumberDcInstances)
```

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

The total number of rings available is 128; therefore, the `NumProcesses` parameter can only be set to a value that meets the constraints described above.

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



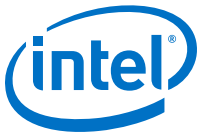
- `NumProcesses` can be set to 10, if `NumberCyInstances` = 2 (consuming 120 rings), with 2 rings for administration, giving a total of 122 (meeting the <128 constraint).
- `NumProcesses` can be set to 20, if `NumberCyInstances` = 1 (consuming 120 rings), with 2 rings for administration, giving a total of 122 (meets the <128 constraint).

5.4 Sample Configuration File (V2)

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

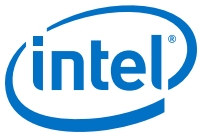
```
############################################################################
#
# @par
# This file is provided under a dual BSD/GPLv2 license.  When using or
#   redistributing this file, you may do so under either license.
#
#   GPL LICENSE SUMMARY
#
#   Copyright(c) 2007-2013 Intel Corporation. All rights reserved.
#
#   This program is free software; you can redistribute it and/or modify
#   it under the terms of version 2 of the GNU General Public License as
#   published by the Free Software Foundation.
#
#   This program is distributed in the hope that it will be useful, but
#   WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
#   General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#   Foundation, Inc., 51 Franklin St - Fifth Floor, Boston, MA 02110-1301 USA.
#   The full GNU General Public License is included in this distribution
#   in the file called LICENSE.GPL.
#
#   Contact Information:
#   Intel Corporation
#
#   BSD LICENSE
#
#   Copyright(c) 2007-2013 Intel Corporation. All rights reserved.
#   All rights reserved.
#
#   Redistribution and use in source and binary forms, with or without
#   modification, are permitted provided that the following conditions
#   are met:
#
#     * Redistributions of source code must retain the above copyright
#       notice, this list of conditions and the following disclaimer.
#     * Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in
#       the documentation and/or other materials provided with the
#       distribution.
#     * Neither the name of Intel Corporation nor the names of its
#       contributors may be used to endorse or promote products derived
#       from this software without specific prior written permission.
#
#   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
#   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
#   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
#   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
#   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
#   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
```

```
#   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
#   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
#   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
#
#  version: QAT1.5.L.1.10.0-65
############################################################################
############################################
# General Section
############################################

[GENERAL]
ServicesEnabled = cy0;cy1

# 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:
```



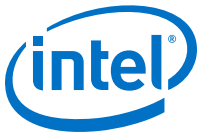
```
# - Name of the logical instance
# - The accelerator associated with this logical
#   instance
# - The core the instance is affinitized to (optional)
#
# Note: Logical instances may not share the same ring, but
#           may share a ring bank.
#
# The format of the logical instances are:
# - For 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 = 2

############################################
# 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 = 2

############################################
# 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: 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
```

5.5 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 and asymmetric cryptography) 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.

  In order to use this functionality, the processes must be started with the icp_sal_userStartMultiProcess function.
- The `LimitDevAccess` parameter has been added. This parameter indicates if the user space processes in the section containing the `LimitDevAccess` parameter are limited to only access instances on a specific device.

6.0 Secure Architecture Considerations

This chapter describes the potential threats identified as part of the secure architecture analysis of the Acceleration Complex within the Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure 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 SoC.

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 SoC device are described.

6.1 Terminology

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

- Threat Categories on page 66
- Attack Mechanism on page 66
- Attacker Privilege on page 67
- Deployment Models on page 67

6.1.1 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

6.1.2 Attack Mechanism

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

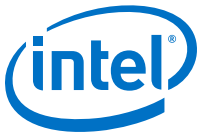


Table 9. Attack Mechanisms and Examples

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.

6.1.3 Attacker Privilege

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

Table 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 SoCto access (read or write) any data to which they would not otherwise have access.
Ability to send packets	In almost all deployments, attackers have the ability to send arbitrary packets from the network (either on LAN or WAN) into the system. It is assumed that threats (for example, contrived packet streams to exploit known vulnerabilities) may arrive in this way.

6.1.4 Deployment Models

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

Table 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

6.2 Threat/Attack Vectors

A thorough analysis has been conducted by considering each of the threat categories, attack mechanisms, attacker privilege levels, and deployment models. As a result, the following threats have been identified. Also described are the steps a user of the SoC 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) and mitigation against those threats are described.

6.2.1 General Mitigation

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

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

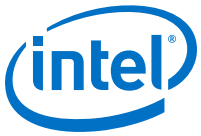
6.2.2 General Threats

General threats include the following:

- DMA on page 68
- Intentional Modification of IA Driver on page 69
- Modification of Intel® QuickAssist Accelerator Firmware on page 69
- Malicious Application Code on page 70
- Contrived Packet Stream on page 70

6.2.2.1 DMA

Threat: The SoC 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 SoC configuration file describes logical instances of acceleration services and the set of hardware rings to be used for each such instance. User processes can ask the kernel driver to map these rings into their address spaces. To access a given device (identified by the number <N> in the filenames below), the user must be granted read/write access to the following files, which may be in /dev or /dev/icp_mux:

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

The recommendation is that these files have the following permissions by 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.

6.2.2.2 Intentional Modification of IA Driver

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

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

6.2.2.3 Modification of Intel® QuickAssist Accelerator Firmware

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

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 68 threat to limit access to this ring.

6.2.2.4 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 68.

6.2.2.5 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:

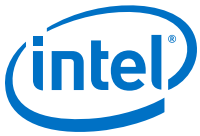
- Sending packets that are encrypted (for example, using IPsec), thereby reducing the availability of these services to legitimate traffic.
- Sending excessively large packets, causing some latency for legitimate packets.
- Sending small packets at a high packet rate, causing extra bandwidth utilization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

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

6.2.3 Threats Against the Cryptographic Service

Threats against the cryptographic service include:

- Reading and Writing of Cryptographic Keys on page 71
- Modification of Public Key Firmware on page 71
- Failure of the Entropy Source for the Random Number Generator on page 71
- Interference Among Users of the Random Number Service on page 71



6.2.3.1 Reading and Writing of Cryptographic Keys

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

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

6.2.3.2 Modification of Public Key Firmware

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

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

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

6.2.3.3 Failure of the Entropy Source for the Random Number Generator

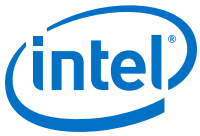
Threat: The SoC has a non-deterministic random bit generator (NRBG, aka True Random Number Generator or TRNG) implemented in silicon that can be used as an entropy source for a deterministic random bit generator (DRBG, aka Pseudo Random Number Generator or PRNG). A failure of the entropy source can lead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

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

6.2.3.4 Interference Among Users of the Random Number Service

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

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



7.0 Supported APIs

The supported APIs are described in two categories:

- Intel® QuickAssist Technology APIs on page 73
- Additional APIs on page 76

7.1 Intel® QuickAssist Technology APIs

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

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

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

7.1.1 Intel® QuickAssist Technology API Limitations

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

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

  - Maximum seed length is 255 bytes
  - Maximum maskLenInBytes is 65528
- For the cryptographic service, SNOW 3G and KASUMI operations are not supported when *CpaCySymPacketType* is set to CPA_CY_SYM_PACKET_TYPE_PARTIAL. The error returned in this case is CPA_STATUS_INVALID_PARAM.
- For the cryptographic service, when using the Deterministic Random Bit Generator (DRBG), only one in-flight request per each instantiated DRBG (that is, per each DRBG session) is allowed. If the user calls the `cpaCyDrbgGen` 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 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.

7.1.2 Data Plane APIs Overview

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
```

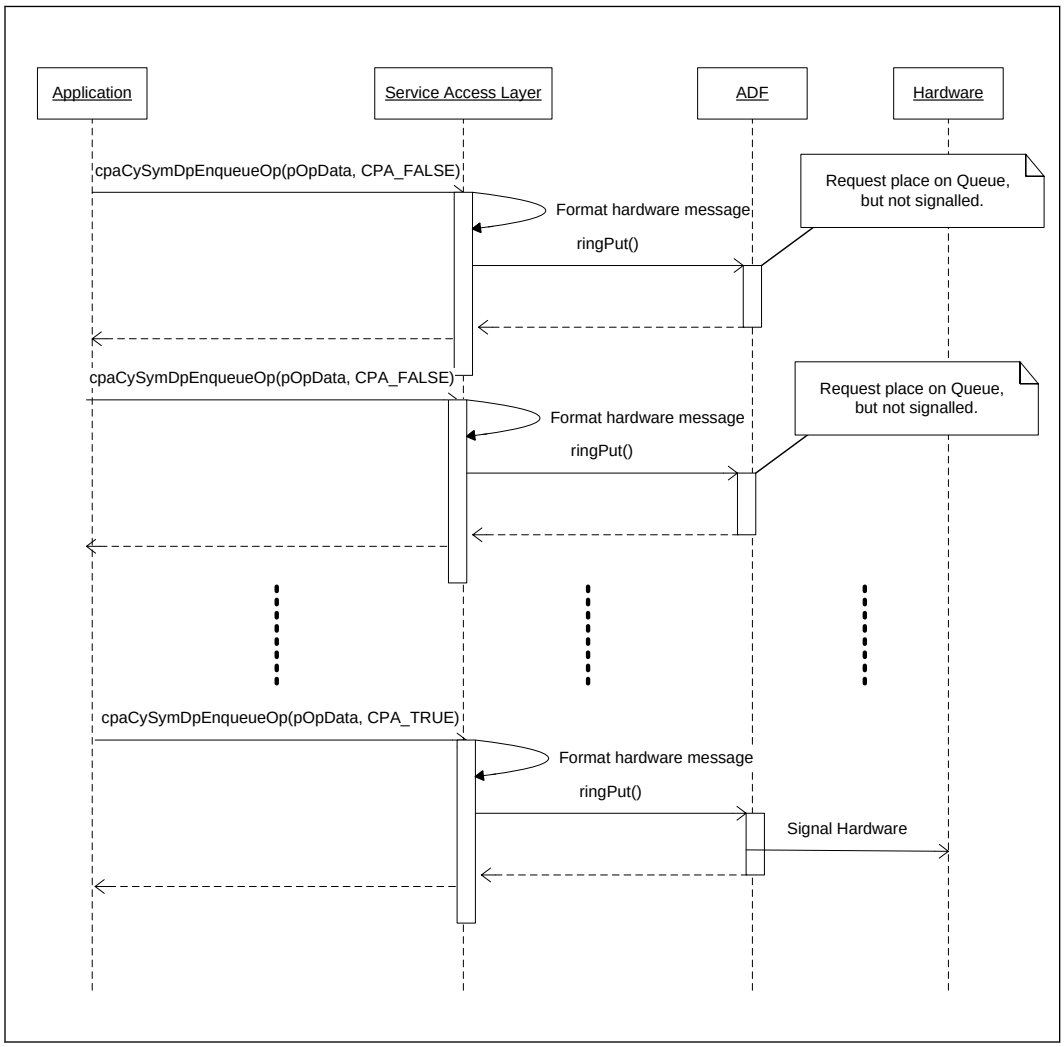
7.1.2.1 IA Cycle Count Reduction When Using Data Plane APIs

From an IA cycle count perspective, the Data Plane APIs are more performant than the traditional APIs (that is, for example, the symmetric cryptographic APIs defined in `$ICP_ROOT/quickassist/include/lac/cpa_cy_sym.h`). The majority of the cycle count reduction is realized by the reduction of supported functionality in the Data Plane APIs and the application of constraints on the calling application (see Usage Constraints on the Data Plane APIs on page 75).

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()` function multiple times with requests to process and the `performOpNow` flag set to CPA_FALSE. Once multiple requests have been enqueued, the `cpaCySymDpEnqueueOp()` 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 14. 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.

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

7.1.2.2 Usage Constraints on the Data Plane APIs

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

- Thread safety is not supported. Each software thread should have access to its own unique instance (`CpaInstanceHandle`) to avoid contention on the hardware rings.
- For performance, polling is supported, as opposed to interrupts (which are comparatively more expensive). Polling functions (see Polling Functions on page 80) 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.

7.1.2.3 Cryptographic API Descriptions

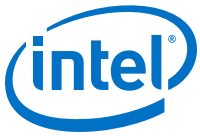
Full descriptions of the Intel® QuickAssist Technology APIs are contained in the *Intel® QuickAssist Technology Cryptographic API Reference Manual*. In addition to the Intel® QuickAssist Technology Data Plane APIs, there are a number of Data Plane Polling APIs that are described in Polling Functions on page 80.

7.2 Additional APIs

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

These APIs are grouped into the following categories:

- Dynamic Instance Allocation Functions on page 77
- Polling Functions on page 80
- Random Number Generation Functions
- User Space Access Configuration Functions on page 84
- Version Information Function on page 89
- User Space Heartbeat Functions on page 87
- Reset Device Function on page 89
- Thread-less APIs on page 90



7.2.1 Dynamic Instance Allocation Functions

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

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

The advantage of using these functions is that the number of cy instances can be changed on-demand at runtime. The disadvantage is that runtime performance is impacted if the number of cy 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 78
- icp_sal_userCyInstancesAlloc on page 78
- icp_sal_userCyFreeInstances on page 79
- icp_sal_userCyGetAvailableNumDynInstancesByDevPkg on page 79
- icp_sal_userCyInstancesAllocByDevPkg on page 80

7.2.1.1 icp_sal_userCyGetAvailableNumDynInstances

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

Syntax

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

Parameters

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

Return Value

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

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

7.2.1.2 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);
```

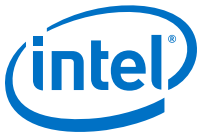
Parameters

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.



Code	Meaning
CPA_STATUS_FAIL	Indicates a failure.

7.2.1.3 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.

7.2.1.4 icp_sal_userCyGetAvailableNumDynInstancesByDevPkg

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

Syntax

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

Parameters

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

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

Return Value

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

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

7.2.1.5 icp_sal_userCyInstancesAllocByDevPkg

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

Syntax

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

Parameters

numCyInstances The number of cy instances to allocate.

*pCyInstances A pointer to the cy instances.

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

Return Value

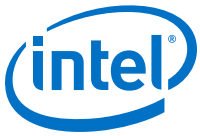
The `icp_sal_userCyInstancesAllocByDevPkg` function returns one of the following codes:

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

7.2.2 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

7.2.2.1 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 c2xxx devices in the system.
bank_number	The number of the memory bank on the c2xxx device that will be polled for response messages. The valid range is 0 to 7.
response_quota	The maximum number of responses to take from the ring in one call.

Return Value

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

Code	Meaning
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.

7.2.2.2 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 c2xxx 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.

7.2.2.3 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 109.

Syntax

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

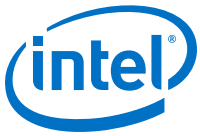
Parameters

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.



Code	Meaning
	Note: A ring is only polled if it contains data.
CPA_STATUS_FAIL	Indicates a failure.

7.2.2.4 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 75 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.

7.2.3 Random Number Generation Functions

Intel® Secure Key, previously code-named Bull Mountain Technology, is the Intel name for the Intel® 64 and IA-32 architectures' instruction RDRAND and its underlying Digital Random Number Generator (DRNG) hardware implementation. Among other things, the DRNG implementation (using the RDRAND instruction) is useful for generating high-quality keys for cryptographic protocols.

For additional details please refer to the Intel® Digital Random Generator (DRNG) Software Implementation Guide.

7.2.3.1 Non-Deterministic Random Bit Generation (NRBG) APIs

The Intel® QuickAssist Non-Deterministic Random Bit Generation APIs are not supported on this platform. These APIs interacted with True Random Number Generator hardware feature on the DH89xxCC product to provide entropy source to the DRBG API. For the Intel® Atom™ Processor C2000, this functionality is provided by the RDRAND instruction.

7.2.3.2 Deterministic Random Bit Generation (DRBG) APIs

The DRBG API implementation was updated to utilize the RDRAND instruction. This was done as a convenience for compatibility with applications that used the DRBG API on the DH89xxCC product and want to run the same application on the SoC. Logic was added to the DRBG APIs to detect and use the RDRAND instruction if supported on the platform.

If the RDRAND instruction is not supported, the DRBG APIs default to the DH89xxXX implementation. Note that the customer application is responsible for registering support functions with the driver. Please refer to the *Intel® Communications Chipset 89xx Series Software Programmer's Guide* for additional information. These functions are described in detail in the section titled "Random Number Generation Functions".

The following table shows the restrictions/key notes that apply when using the DRBG API via RDRAND.

Function	Description
cpaCyDrbgSessionInit	Only 1 instantiation of DRBG supported in the system. Multiple DRBG sessions may be present but they use the same instantiated DRBG. Security strength of 128 bits or less. No support for personalization string. No support for prediction resistance.
cpaCyDrbgReseed	Not supported because reseeding occurs autonomously.
cpaCyDrbgGen	Security strength of 128 bits or less. No support for prediction resistance. No support for additional Input. Reseeding occurs autonomously.

For additional DRBG API details, please refer to the *Intel® QuickAssist Technology Cryptographic API Reference Manual* and *Intel® QuickAssist Technology API Programmer's Guide*.

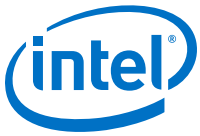
7.2.4 User Space Access Configuration Functions

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

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

The user space access configuration functions include:

- icp_sal_userStartMultiProcess
- icp_sal_userStart



- icp_sal_userStop

7.2.4.1 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

7.2.4.2 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.

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

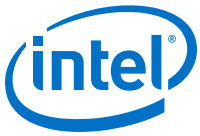
The `icp_sal_userStartMultiProcess` function returns one of the following codes:

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.

7.2.4.2.1 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)`. If the application spawns more child processes, the first N processes that call `icp_sal_userStartMultiProcess("SSL", CPA_TRUE)` start successfully with access to the accelerator. All subsequent calls start successfully but will not have access to the accelerator. In this case, calls to `cpaCyGetNumInstances()` return zero. If any of the N running processes finish their work and call `icp_sal_userStop()` (or if a subprocess terminates non-gracefully), another subprocess can call `icp_sal_userStartMultiProcess("SSL", CPA_TRUE)` and it will succeed.

7.2.4.3 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

7.2.5 User Space Heartbeat Functions

These functions allow the user space application to check the status of the firmware/ hardware of the acceleration 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 88
- icp_sal_check_all_devices on page 88

7.2.5.1 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.
CPA_STATUS_FAIL	Operation failed.

Notes

None

7.2.5.2 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);
```

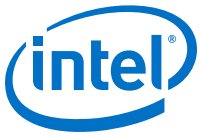
Parameters

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.



7.2.6 Version Information Function

A function that allows the retrieval of version information related to the software and hardware being used.

The version information function definition is located in: `$ICP_ROOT/quickassist/lookaside/access_layer/include/icp_sal_versions.h`.

There is only one version information function, that is, `icp_sal_getDevVersionInfo`.

7.2.6.1 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.

7.2.7 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 37 for details of the steps on a device reset. The device can also be reset using the adf_ctl utility, e.g., by calling `adf_ctl icp_dev0 reset`.

7.2.7.1 icp_sal_reset_device

Resets the device.

Syntax

```
CpaStatus icp_sal_reset_device ( Cpa32U accelid);
```

Parameters

accelid The device number.

Return Value

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

Code	Meaning
CPA_STATUS_SUCCESS	Successful operation.
CPA_STATUS_FAIL	Indicates a failure.

7.2.8 Thread-less APIs

These APIs can be used in the User Space Application when the driver is built with the ICP_WITHOUT_THREAD flag. See Thread-less Mode on page 40 for details.

The Thread-less API functions include:

- icp_sal_poll_device_events on page 90
- icp_sal_find_new_devices on page 91

7.2.8.1 icp_sal_poll_device_events

This reads any pending device events from icp_dev%d_csr (see Driver Threading Model on page 39) and forwards to interested subsystems.

Syntax

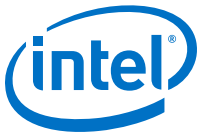
```
CpaStatus CpaStatus icp_sal_poll_device_events(void) ( Cpa32U
accelid);
```

Parameters

none

Return Value

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



Code	Meaning
CPA_STATUS_SUCCESS	Successful operation.
CPA_STATUS_FAIL	Indicates a failure.

7.2.8.2 icp_sal_find_new_devices

This tries to connect to any available devices that the kernel driver has brought up and initialized for use in user space process.

Syntax

```
CpaStatus CpaStatus icp_sal_find_new_devices(void) ( Cpa32U
accelid);
```

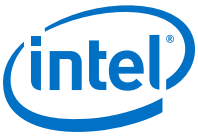
Parameters

none

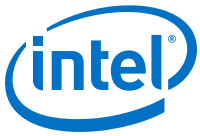
Return Value

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

Code	Meaning
CPA_STATUS_SUCCESS	Successful operation.
CPA_STATUS_FAIL	Indicates a failure.



Part 3: Applications and Usage Models



8.0 Application Usage Guidelines

This chapter provides some usage guidelines and identifies some of the applications to which the platforms described in this manual are ideally suited.

Note: The usage information provided in this section relates to the original configuration file format. Much of the information is still appropriate when using the newer (default) version of the configuration file.

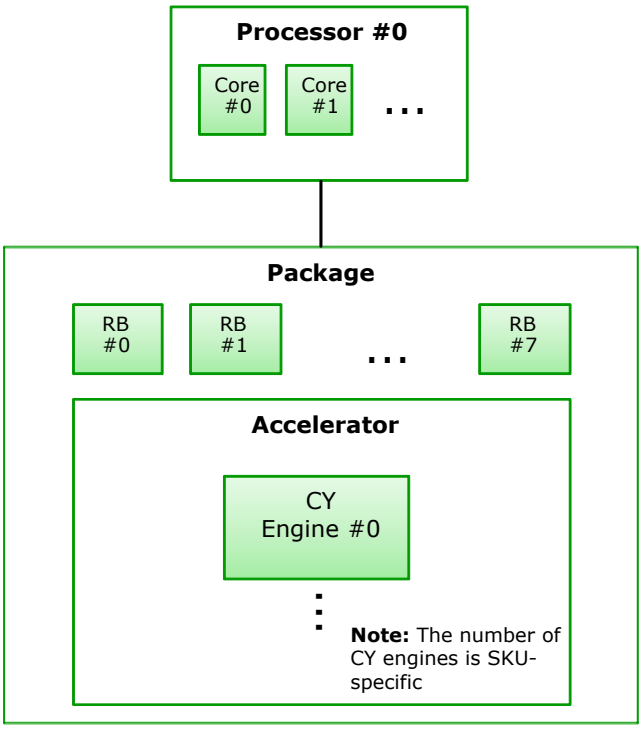
8.1 Mapping Service Instances to Hardware Accelerators on the SoC

The SoC contains zero or one accelerator depending on the device variant being used. There are two cryptography engines within the accelerator. There is one engine for low (SKU 2) and mid SKU (SKU 3), and two for the high end.

A set of 16 ring banks provide the communication mechanism between a processor and the acceleration complex on SoC device. Each ring bank contains 16 individual rings for communication. The following figure shows the relationship between processors, 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 15. Processor and SoC Device Components

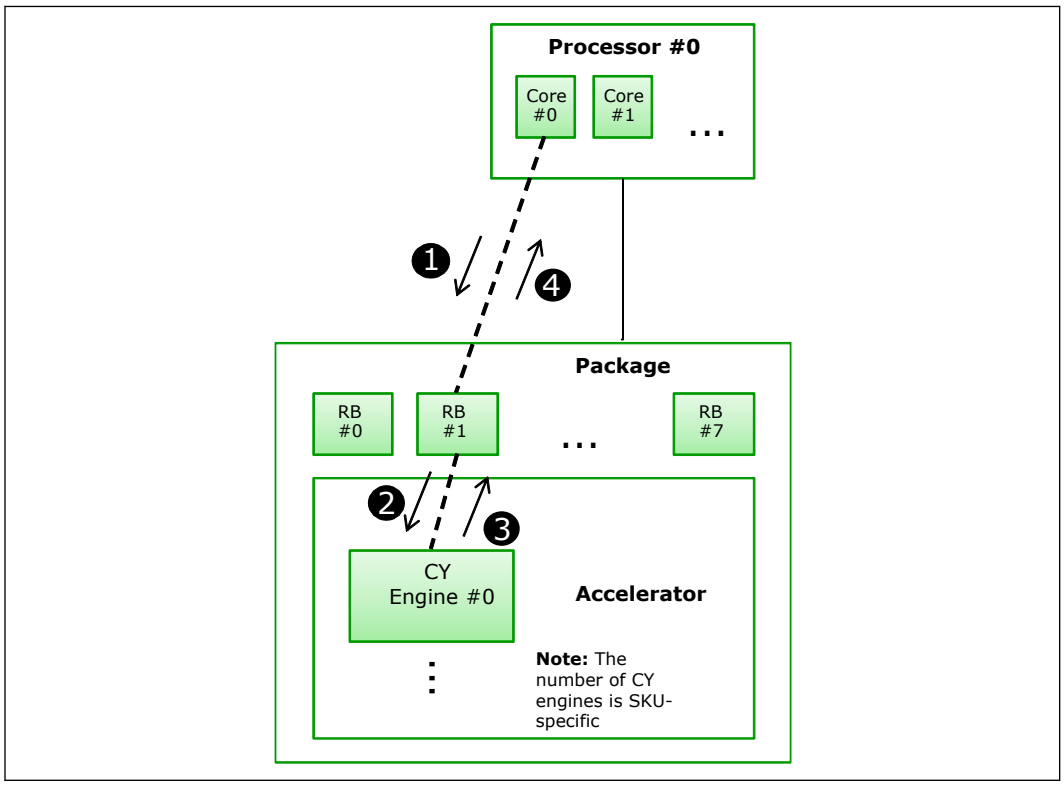


8.1.1 Processor and SoC Device Communication

An acceleration service uses different rings for request and response messages and for different priorities (currently for symmetric cryptography only). Communication between the processor and SoC 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 SoC device uses a read (get) operation to retrieve the request from the request ring.
3. Once the operation has been performed, the SoC 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 16. SoC Device Communication



8.1.2 Service Instances and Interaction with the Hardware

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.

In general, a service instance uses a pair of rings, one for requests and one for responses. For cryptographic instances, separate request/response pairs are used for the following:

- Symmetric low priority
- Symmetric high priority
- Public key cryptography requests/responses

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

Table 12. Service Instance Attributes

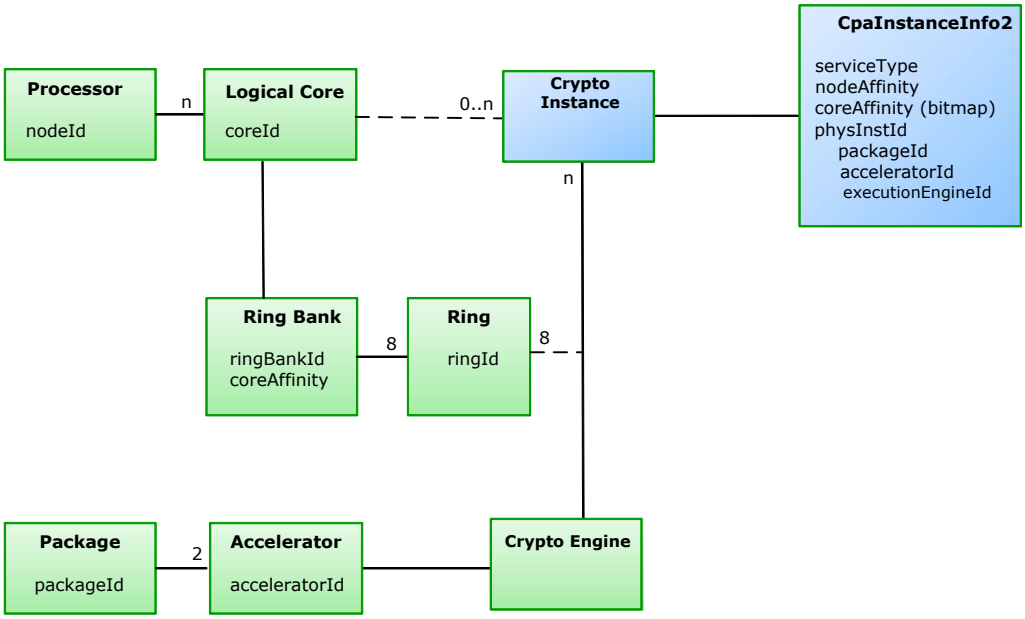
Member	Sub-field	Description
physInstId	acceleratorId	Identifies the accelerator within the SoC
physInstId	executionEngineId	Identifies the engine (slice) within the accelerator

continued...

Member	Sub-field	Description
nodeAffinity	N/A	Identifies the processor node/socket to which the SoC is physically connected (relevant in NUMA configurations)
coreAffinity	N/A	Identifies the core(s) to which interrupts (if enabled) are affinitized (Bitmap)
isPolled	N/A	For Kernel space: • IsPoll = 0 (interrupt mode) • IsPoll = 1 (poll mode) For User space: • IsPoll = 0 (interrupt mode) • IsPoll = 1 (poll mode)

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

Figure 17. Service Instance Attributes and Hardware Components



8.1.3 Service Instance Configuration

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

Note: The following example uses the earlier configuration file format, which continues to be supported.

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

Figure 18. Service Instance Configuration

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


# Crypto - user space instance #0
Cy0Name = "proc0.0"
Cy0AcceleratorNumber = 0 2
Cy0ExecutionEngine = 0 3
Cy0BankNumber = 1 4
Cy0IsPolled = 1

Cy0RingAsymTx = 2 5
Cy0RingAsymRx = 3
Cy0RingSymTxHi = 4 6
Cy0RingSymTxLo = 5
Cy0RingSymRxHi = 6 7
Cy0RingSymRxLo = 7
```

In the previous figure, the meaning of each numbered item is explained as follows:

1. Each named address domain (one domain for the kernel, any number of user space process domains) has its own service instances.
2. Identifies the accelerator.
3. Identified the accelerator engine.
4. Identifies the ring bank to be used by the instances (which has a core affinity). See the configuration file snipped below for an example of core affinity association.
5. Asymmetric (public key), request (Tx) and response (Rx) rings.
6. Symmetric (bulk) crypto, low (normal) and high priority request (Tx) rings; *Cy0RingSymTxHi* is for the high priority requests and *Cy0RingSymTxLo* is for normal priority requests.
7. Symmetric (bulk) crypto, low (normal) and high priority response (Rx) rings; *Cy0RingSymRxHi* is for the high priority responses and *Cy0RingSymRxLo* is for normal priority responses.

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

The application is responsible for load balancing/spreading:

- Across engines within a SoC device

To get the maximum performance from the hardware, there needs to be at least as many service instances as engines, that is:

- Four for Cryptography.

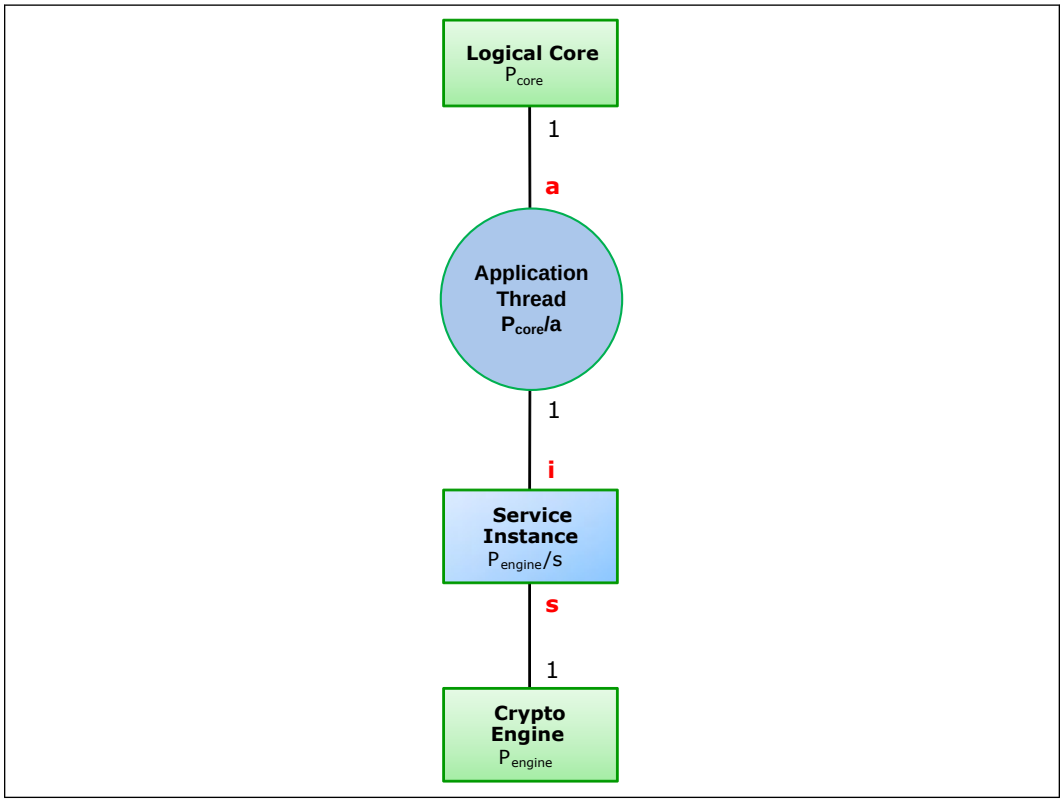
In the simplest case, load balancing is done through configuration. This applies when each engine has more capacity than required by an lcore. Each lcore uses exactly one service instance. Different lcores use different service instances, which map to different service engines. The load is balanced by spreading the traffic across lcores.

If a hardware engine has more capacity than that required by one instance, then multiple instances can share an engine. If a hardware engine has less capacity than required by one instance, then a core/process can talk to multiple instances.

Each core (physical or logical) has a certain application performance capacity (P_{core}). This depends on the core frequency, number of IA cycles per packet, packet size/mix, protocols and so on. Each (physical) service engine has a certain level of service performance (P_{engine}). This may depend on the SKU, cryptography algorithms, packet size/mix and so on.

The following figure shows the relationship between cores, application threads, service instances and cryptographic engines.

Figure 19. Entities and Relationships for Load Balancing



The goal is to balance the performance of the cores and the service engines. Expressed mathematically, choose *a*, *i* and *s* (see figure), such that:

$P_{core}/a \sim= (P_{engine})*i/s$

Note: Performance capacity of a core may be measured as throughput at a certain packet size, mix of sizes, protocols and so on. Performance capacity of a service engine can be measured as throughput at a certain packet size, mix of sizes, algorithms and so on.

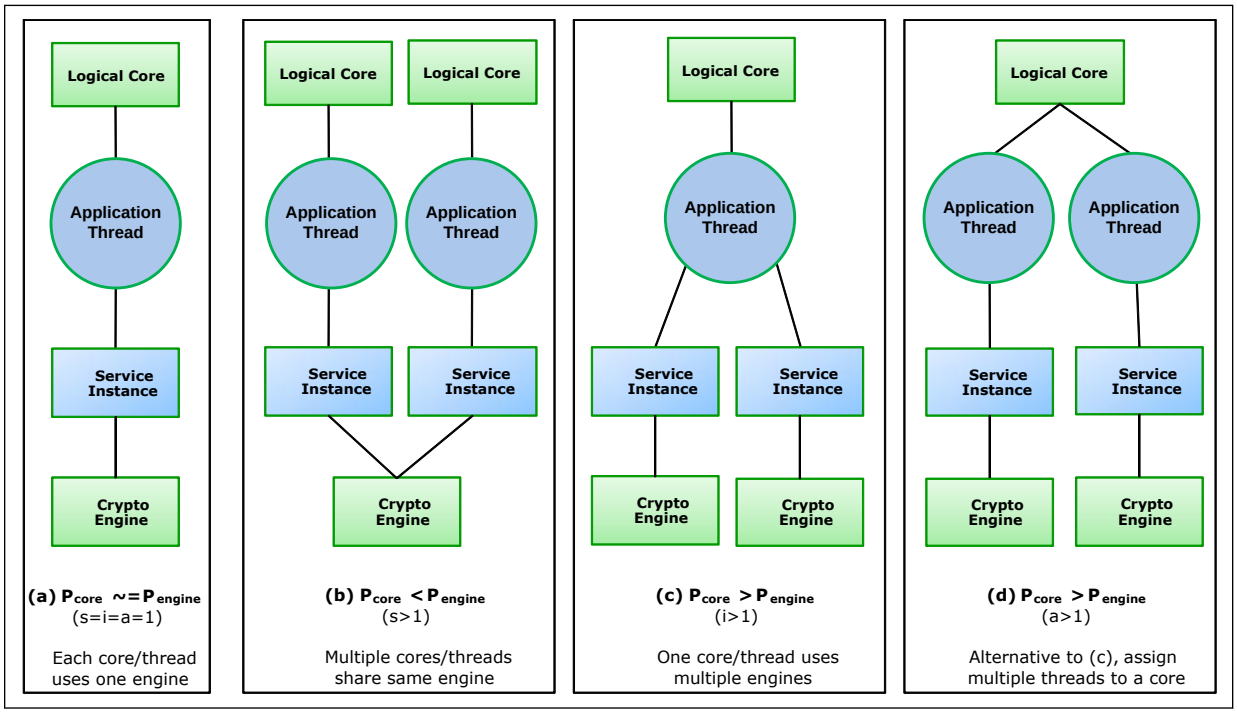
The following figure shows four different load balancing scenarios:

- Case (a) - The simplest case. Load balancing is done by spreading traffic across cores, and then each core talks to exactly one engine.
- Case (b) - When the engines have more capacity than the cores/threads need, then an engine can be mapped into multiple such cores/threads.

When the cores need more capacity than one engine can supply, then multiple engines must be mapped into a single lcore. There are at least two ways to do this as indicated in case (c) and (d) following.

- Case (c) - Each thread talks to multiple service instances. This requires the application code to change, in that the application must know about multiple instances and load balance across them.
- Case (d) - Multiple threads can be assigned to the same lcore. This moves the responsibility for load balancing to the OS or whatever is managing the threads.

Figure 20. Load Balancing Scenarios



In all cases, except Case (c), the code remains unchanged. Each thread talks to exactly one service instance. This makes it easier to port applications to different platforms with different numbers/frequencies of cores, different numbers/ SKU numbers and so on.

8.2 Cryptography Applications

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

- Virtual Private Networks (VPNs, both IPsec and SSL). Both symmetric and public key cryptography can be offloaded for bulk transfer and key exchange (IKE, SSL handshakes and so on). See IPsec and SSL VPNs on page 100 for more information.
- Encrypted Storage. See Encrypted Storage on page 100 for more information.
- Web Proxy Appliances. See Web Proxy Appliances on page 101.

See also the Accelerating a Security Appliance white paper. This was first written to support the Intel® EP80579 Integrated Processor with Intel® QuickAssist Technology. Many of the concepts and ideas are applicable to the platforms described in this manual also.

8.2.1 IPsec and SSL VPNs

Virtual Private Networks (VPNs) allow for private networks to be established over the public internet by providing confidentiality, integrity and authentication using cryptography. VPN functionality can be provided by a standalone security gateway box at the boundary between the trusted and untrusted networks. It is also commonly combined with other networking and security functionality in a security appliance, or even in standard routers.

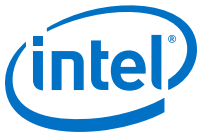
VPNs are typically based on one of two cryptographic protocols, either IPsec or DTLS. Each has its advantages and disadvantages.

One of the most compute-intensive aspects of a VPN is the cryptographic processing required to encrypt/decrypt traffic for confidentiality, to perform cryptographic hash functionality for authentication and to perform public key cryptography, based on modular exponentiation of large numbers or elliptic curve cryptography as part of key negotiation and exchange. The Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure 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 SoCoffers 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 SoC's cryptographic services to be accessed through open source cryptographic frameworks, such as the Linux kernel crypto framework/API (also known as the *scatterlist* API) and OpenSSL's libcrypto (through its EVP API). This facilitates ease of integration with certain open source implementations of protocol stacks, such as the Linux kernel's native IPsec stack (called NETKEY) or with OpenVPN (an open source SSL VPN implementation).

8.2.2 Encrypted Storage

In recent years, cases of lost laptops containing sensitive information have made the headlines all too frequently. Full disk encryption has become a standard procedure for many corporate PCs. Safe-guarding critical data however is not just a necessity in the client space, it is also a necessity in the data center.



Enterprise-class storage appliances achieve throughput rates in excess of 50 Gbps. Several high-profile cases of data theft have triggered updates to government regulations and industry standards. These regulations/standards now require protection of data-at-rest for applications involving sensitive data such as medical and financial records, typically using strong encryption. The high computational cost of adding security to storage appliances makes offload solutions an attractive value proposition.

Several complimentary standards for the security of data-at-rest exist, which when combined with traditional network security protocols, such as IPsec or SSL/TLS, provide an end-to-end secure storage solution, even for data-in-flight.

The IEEE Security in Storage working group is developing the IEEE 1619 series of standards that deal with cipher algorithms for disk and tape storage devices (AES in CCM and GCM modes). The cryptographic acceleration services of platforms that use the are ideally suited for secure long-term storage solutions implementing the IEEE 1619.1 standard, by providing acceleration of the AES-256 cipher in CBC, CCM, and GCM modes and HMAC authentication using SHA-1, SHA-256 and SHA-512 hashes.

The Trusted Computing Group's (TCG) Storage Working Group does not prescribe a particular set of algorithms for the disk encryption. Instead, it defines several Storage Subsystem Classes (SSC) for various usage models, which define services such as enrollment and connection, protected storage (an extension of TPM), locking, logging, cryptographic services, authorization, and firmware updates. The cryptographic acceleration services of the platform can help by providing the highest level of security for authenticating the host to trusted peripherals implementing the TCG storage standards.

8.2.3 Web Proxy Appliances

Historically, Web Proxy appliances have evolved to present a public or intermediary interface for clients seeking resources from other servers, providing services such as web page caching and load balancing. These appliances are located at the edge of the network, typically at network gateways. Due to their centralized presence in the network, Web Proxy appliances today (referred to with a number of different names, such as Application Delivery Controllers, Reverse Proxy, and so on) have become a collection of services that include:

- Application Load Balancing (L4-L7)
- SSL Acceleration
- WAN Acceleration
- Caching
- Traffic Management
- Web Application Firewall

SSL and WAN acceleration have become common place capabilities of the Web Proxy appliance, requiring compute intensive algorithms for cryptography (SSL). Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure 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 SoC). With the prominence of Web Proxy appliances in typical networks, this use case has applications from cloud computing to small web server deployments.

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.

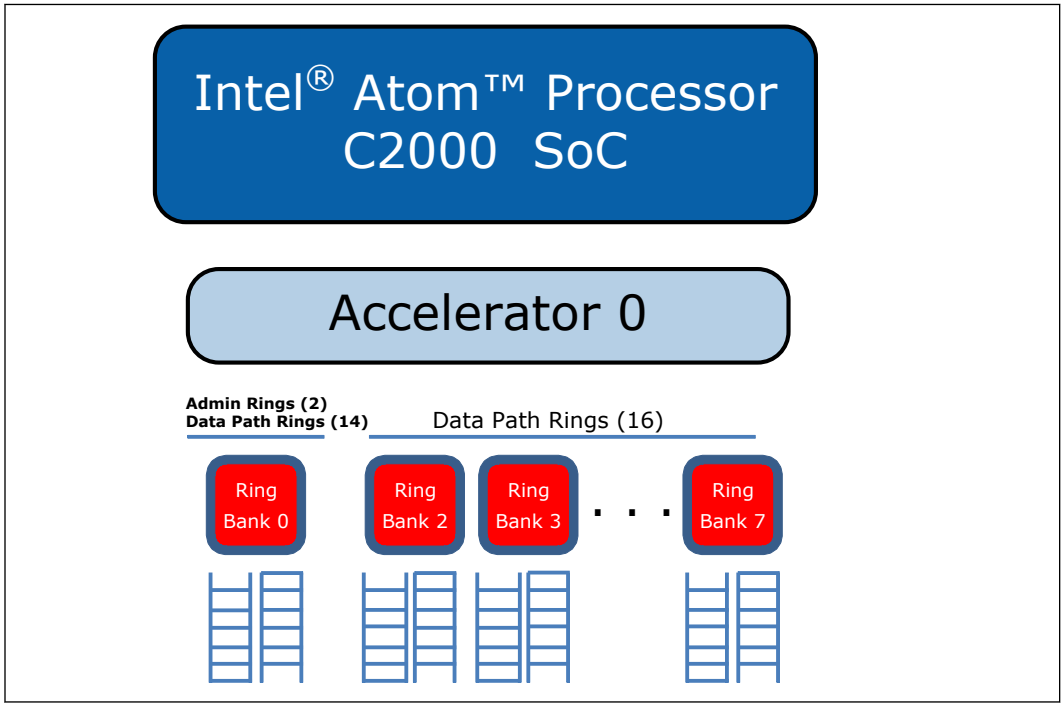
A.1 Configuration File Overview

There is a single configuration file for each Intel® Atom™ Processor C2000 Product Family for Communications Infrastructure device. The configuration file always contains two accelerator subsections. The significance of these subsections depends on the number of accelerators in the SoC device as defined by the model number:

- If there are no accelerators in the device, the information in both accelerator subsections is not relevant and can be ignored.

Each accelerator has eight independent ring banks - the communication mechanism between the Acceleration software and the hardware. Each ring bank has an interrupt that can be directed to a specific Intel® architecture core. Each ring bank has 16 rings (hardware assisted queues). This hierarchy is shown in the following figure.

Figure 21. Ring Banks



Note: Depending on the SKU number, the SoC device may also contain no accelerators.

The configuration file is split into three (or more) sections: General, Hardware Access Ring Bank Configuration, and one or more Logical Instance sections.

- **General** - includes parameters that allow the user to:
  - Specify which services are enabled.
  - Configure the settings for the services.

  Additional details are included in General Parameters on page 104.
- **Hardware Access Ring Bank Configuration** - includes parameters that allow the user to:
  - Enable and configure interrupt coalescing.
  - Direct an MSI-x interrupt for a given ring bank to a specified Intel® architecture core, assuming that the OS supports MSI-X interrupts.

  Additional details are included in [AcceleratorX] Section on page 106.
- **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 108.

A sample configuration file, targeted at a high-end IPsec box, is included in Sample Configuration File (V1) on page 111.

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 51

Table 13. General Parameters - Earlier File Format

Parameter	Description	Default	Range
ServicesEnabled	Defines the service(s) available (cryptographic [cyX]).	cy0	cyX *Note:* Depending on the SKU, X can be 0 or 1, which identifies one of two available cryptographic engines. *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.
NumberOfWirelessProcs	Defines the number wireless processes.	0	0 to 2
Firmware_UofPath	Name of the Microcode (UCode) Object File (UOF) firmware.	uof_firmware.bin	uof_firmware.bin
Firmware_MmpPath	Name of the Modular Math Processor (MMP) firmware.	mmp_firmware.bin	mmp_firmware.bin
Note: "Default" denotes the value in the configuration file when shipped.			

A.2.2 QAT Parameters

The following table describes accelerator-specific parameters.

Note: In the following parameters, beginning AccelX..., the X can be 0 or 1 representing the accelerator number.

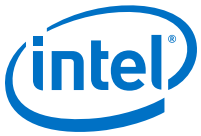


Table 14. QAT Parameters - Earlier File Format

Parameter	Description	Default	Range
AccelXAdminBankNumber	Specifies the bank number for administration request/response rings on accelerator X, where X can be 0 or 1.	0	0 to 7
AccelXAcceleratorNumber	Specifies the accelerator number for administration request/response rings for accelerator X, where X can be 0 or 1.	0	0 or 1
AccelXAdminTx	Specifies the ring number of the administration request ring for accelerator X, where X can be 0 or 1.	0	0
AccelXAdminRx	Specifies the ring number of the administration response ring for accelerator X, where X can be 0 or 1.	1	1
Note: "Default" denotes the value in the configuration file when shipped.			

A.2.3 Statistics Parameters

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

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

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

- Calling the appropriate Intel® QuickAssist Technology API function. For example, `cpaCySymQueryStats` or `cpaCySymQueryStats64` for symmetric cryptography. See the *Intel® QuickAssist Technology Cryptographic API Reference Manual* for more information about these functions.
- For kernel space instances, looking at entries in the `/proc/icp_c2xxx_devX` directory, where X is the device number. For example, `/proc/icp_c2xxx_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 40 for more information.

Table 15. 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
			continued...

Parameter	Description	Default	Range
statsKeyGen	Enables/disables statistics for the Key Generation algorithm.	1	1 or 0
statsLn	Enables/disables statistics for the Large Number generator.	1	1 or 0
statsPrime	Enables/disables statistics for the Prime Number detector.	1	1 or 0
statsRsa	Enables/disables statistics for the RSA algorithm.	1	1 or 0
statsSym	Enables/disables statistics for symmetric ciphers.	1	1 or 0
Note: "Default" denotes the value in the configuration file when shipped. A value of 1 indicates "enabled"; a value of 0 indicates "disabled".			

A.3 [AcceleratorX] Section

Note: An SoC device may contain 0 or 1 accelerator depending on the model number. In the configuration file, there is an [AcceleratorX] section for each accelerator.

The [AcceleratorX] section of the configuration file contains interrupt coalescing and core affinity parameters.

A.3.1 Interrupt Coalescing Parameters

For each accelerator, the interrupt coalescing parameters in the following table can be configured.

Table 16. Interrupt Coalescing Parameters - Earlier File Format

Parameter	Description	Default	Range
BankXInterruptCoalescingEnabled	Specifies if interrupt coalescing is enabled for ring bank X, where X is in the range 0 to 7.	1	0 or 1
BankXInterruptCoalescingTimerNs	Specifies the coalescing time, in nanoseconds (ns), for ring bank X, where X is in the range 0 to 7. *Note:* If a value outside the range is set, the default value is used.	10000	500 to 1048575
BankXInterruptCoalescingNumResponses	Specifies the number of responses that need to arrive from hardware before the interrupt is triggered. It can be used to maximize throughput or adjust throughput latency ratio.	0 (disable)	0 to 248
Note: "Default" denotes the value in the configuration file when shipped.			

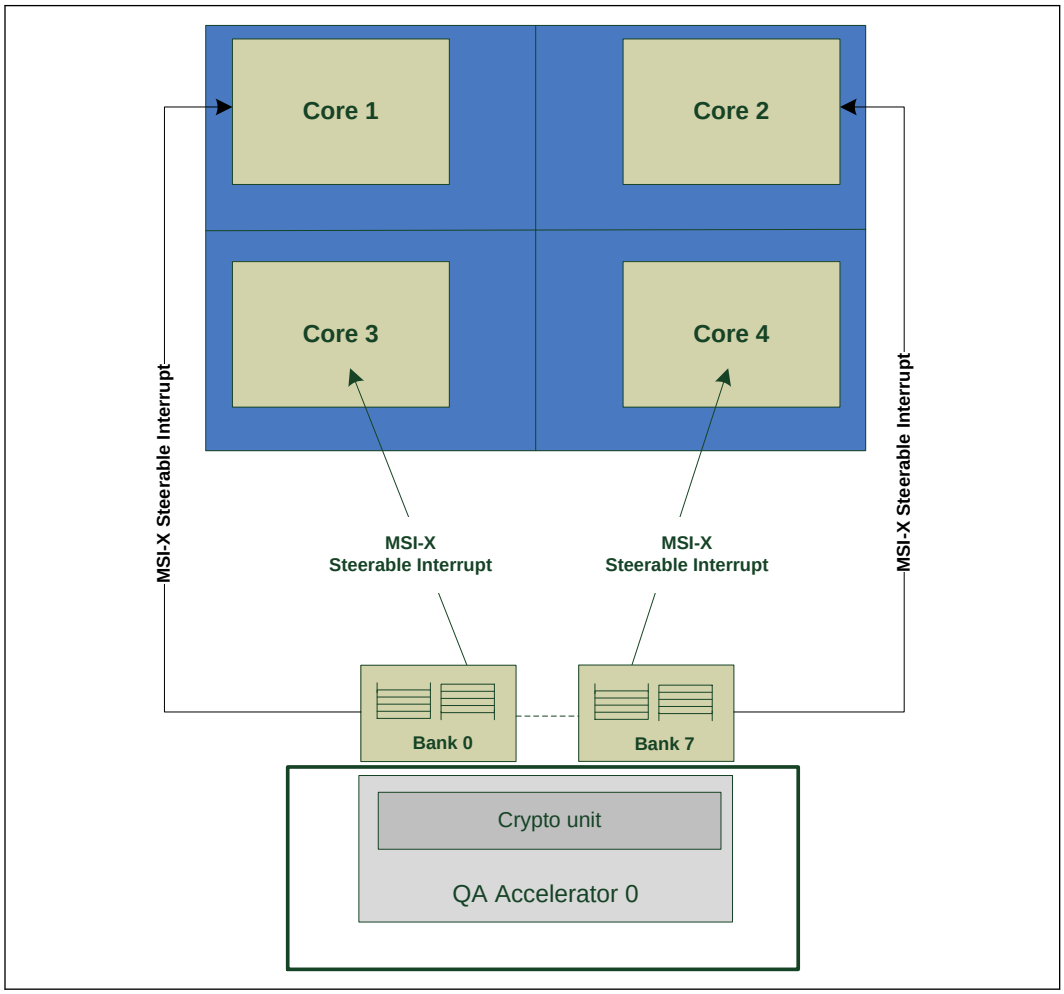
A.3.2 Affinity Parameters

To use core affinity, it is necessary to disable the `irqbalancer` service using the following command issued from an account with root privileges:

```
# service irqbalance stop
```

Each accelerator has eight ring banks (0 to 7). If the OS supports MSI-X interrupts, each ring bank has a steerable MSI-X interrupt that may be affinitized to a particular node/core as shown in the following figure.

Figure 22. Ring Bank Affinity to Core for MSI-X Interrupts



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

Table 17. Ring Bank Affinity Parameters

Parameter	Description	Default	Range
BankXCoreIDAffinity	Defines core affinity for ring bank X, where X is in the range 0 to 7.	0	0 to cpumax-1

continued...

Parameter	Description	Default	Range
			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 22 and Logical Instances on page 16 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 85, passing the xxxxx string during process initialization. When the user space process is finished, it must call the function icp_sal_userStop on page 87 to free resources. See User Space Access Configuration Functions on page 84 for more information.

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

- The name of the logical instance
- The accelerator associated with this logical instance
- The ring bank associated with this logical instance
- The response mode associated with this logical instance (0 for IRQ, 1 for Polled)
- The ring for receiving and the ring for transmitting
- The number of concurrent requests supported by a pair of rings on this instance (Tx and Rx).

  Note: This number affects the amount of memory allocated by the driver. Also, coalescing that is based on the number of responses is only enabled if: 1) Time-based coalescing is enabled, 2) The number of concurrent requests = 512256 (ring size = 16 KB) and 3) Bank<n>InterruptCoalescingNumResponses != 0.

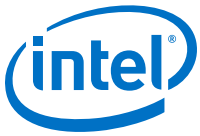
Note: Logical instances may not share the same rings, but may share a ring bank.

A.4.1 [KERNEL] Section

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

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

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



Parameter	Description	Default	Range
NumberCyInstances	Specifies the number of cryptographic instances. *Note:* Depends on the number of allocations to other services.	2	0 to 32
Note: "Default" denotes the value in the configuration file when shipped.			

A.4.1.1 Cryptographic Logical Instance Parameters

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

Table 18. Cryptographic Logical Instance Parameters - Earlier File Format

Parameter	Description	Default	Range
CyXName	Specifies the name of cryptographic instance number X.	IPSec0	String (max. 64 characters)
CyXAcceleratorNumber	Specifies the accelerator number that the cryptographic instance number X is assigned to.	0	0 or 1
CyXBankNumber	Specifies the bank number of the cryptographic instance number X.	0 for kernel space instances 1 for user space instances	0 to 8
CyXExecutionEngine	Specifies the engine that cryptographic instance number X executes on.	0	0 or 1 (depending on the SKU)
CyXIsPolled	Specifies if cryptographic instance number X works in poll mode or IRQ mode.	0 for kernel space instances 1 for user space instances	For instance in the kernel space: 0 (interrupt mode) 1 (poll mode) For instance in the user space: 0 (interrupt mode) 1 (poll mode)
CyXNumConcurrentSymRequests	Specifies the number of cryptographic concurrent 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.	2 for kernel space instances 0 for user space instances	Even number in range: 0 to 14
CyXRingAsymRx	Specifies the asymmetric response ring number for cryptographic instance number X.	3 for kernel space instances	Odd number in range: 1 to 15

continued...

Parameter	Description	Default	Range
		1 for user space instances	
	Specifies the symmetric request ring number for cryptographic instance number X for high priority messages.	4 for kernel space instances 2 for user space instances	Even number in range: 0 to 14
CyXRingSymTxLo	Specifies the symmetric request ring number for cryptographic instance number X for low priority messages.	5 for kernel space instances 3 for user space instances	Even number in range: 0 to 14
CyXRingSymRxHi	Specifies the symmetric response ring number for cryptographic instance number X for high priority messages.	6 for kernel space instances 4 for user space instances	Odd number in range: 0 to 15
CyXRingSymRxHi	Specifies the symmetric response ring number for cryptographic instance number X for low priority messages.	7 for kernel space instances 5 for user space instances	Odd number in range: 1 to 15
Note: "Default" denotes the value in the configuration file when shipped.			

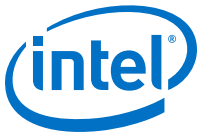
A.4.2 User Process Instance [xxxxx] Sections

In each [xxxxx] section of the configuration file, information about the number and type of user process instances can be defined.

The parameters in the following table specify the number of user process instances for each service.

Parameter	Description	Default	Range
NumberCyInstances	Specifies the number of cryptographic instances. *Note:* Depends on the number of allocations to other services.	0	0 to 32
Note: "Default" denotes the value in the configuration file when shipped.			

Parameters for each user process instance can also be defined. The parameters that can be included for each specific user process instance are similar to those in the Logical Instances Section on page 108.

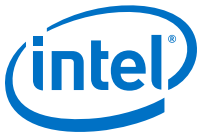


A.5 Sample Configuration File (V1)

The following sample configuration file is intended for a high-end IPsec box.

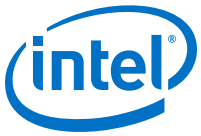
```
##############################################################################
#
# @par
# This file is provided under a dual BSD/GPLv2 license.  When using or
#   redistributing this file, you may do so under either license.
#
#   GPL LICENSE SUMMARY
#
#   Copyright(c) 2007-2013 Intel Corporation. All rights reserved.
#
#   This program is free software; you can redistribute it and/or modify
#   it under the terms of version 2 of the GNU General Public License as
#   published by the Free Software Foundation.
#
#   This program is distributed in the hope that it will be useful, but
#   WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
#   General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#   Foundation, Inc., 51 Franklin St - Fifth Floor, Boston, MA 02110-1301 USA.
#   The full GNU General Public License is included in this distribution
#   in the file called LICENSE.GPL.
#
#   Contact Information:
#   Intel Corporation
#
#   BSD LICENSE
#
#   Copyright(c) 2007-2013 Intel Corporation. All rights reserved.
#   All rights reserved.
#
#   Redistribution and use in source and binary forms, with or without
#   modification, are permitted provided that the following conditions
#   are met:
#
#     * Redistributions of source code must retain the above copyright
#       notice, this list of conditions and the following disclaimer.
#     * Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in
#       the documentation and/or other materials provided with the
#       distribution.
#     * Neither the name of Intel Corporation nor the names of its
#       contributors may be used to endorse or promote products derived
#       from this software without specific prior written permission.
#
#   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
#   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
#   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
#   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
#   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
#   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
#   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
#   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
#   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
#
```

```
#  version: QAT1.5.L.1.10.0-65
############################################################################
#############################################################
#
# This file is the configuration for a single c2xxx_qa
# device.
#
# Each device has one accelerator.
# The accelerator has 8 independent ring banks.
# - The interrupt for each can be directed to a
#   specific core.
# Each ring bank as 16 rings (hardware assisted queues).
#
#############################################################

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

[GENERAL]
ServicesEnabled = cy0;cy1

# Look Aside Cryptographic Configuration
cyHmacAuthMode = 1

# Number of Wireless Processes
NumberOfWirelessProcs = 0

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

# QAT Parameters
Accel0AdminBankNumber = 0
Accel0AcceleratorNumber = 0
Accel0AdminTx = 0
Accel0AdminRx = 1

#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

################################################
#
# Hardware Access Ring Bank Configuration
# Each Accelerator has 8 ring banks (0-7)
# If the OS supports MSI-X, each ring bank has an
# steerable MSI-x interrupt which may be
# affinitized to a particular node/core.
#
################################################


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

Bank1InterruptCoalescingEnabled = 1
```



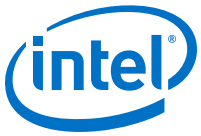
```
Bank1InterruptCoalescingTimerNs = 10000
Bank1CoreIDAffinity = 2
Bank1InterruptCoalescingNumResponses = 0

Bank2InterruptCoalescingEnabled = 1
Bank2InterruptCoalescingTimerNs = 10000
Bank2CoreIDAffinity = 4
Bank2InterruptCoalescingNumResponses = 0

Bank3InterruptCoalescingEnabled = 1
Bank3InterruptCoalescingTimerNs = 10000
Bank3CoreIDAffinity = 6
Bank3InterruptCoalescingNumResponses = 0

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

Bank5InterruptCoalescingEnabled = 1
Bank5InterruptCoalescingTimerNs = 10000
Bank5CoreIDAffinity = 2
Bank5InterruptCoalescingNumResponses = 0

Bank6InterruptCoalescingEnabled = 1
Bank6InterruptCoalescingTimerNs = 10000
Bank6CoreIDAffinity = 4
Bank6InterruptCoalescingNumResponses = 0

Bank7InterruptCoalescingEnabled = 1
Bank7InterruptCoalescingTimerNs = 10000
Bank7CoreIDAffinity = 6
Bank7InterruptCoalescingNumResponses = 0


##########################################################
#
# Logical Instances Section
# A logical instance allows each address domain
# (kernel space and individual user space processes)
# to configure rings (i.e. hardware assisted queues)
# to be used by that address domain and to define the
# behavior of that ring.
#
# The address domains are in the following format
# - For kernel address domains
#       [KERNEL]
# - For user process address domains
#   [xxxxx]
#   Where xxxxx may be any ascii value which uniquely identifies
#   the user mode process.
#   To allow the driver correctly configure the
#   logical instances associated with this user process,
#   the process must call the icp_sal_userStart(...)
#   passing the xxxxx string during process initialisation.
#   When the user space process is finish it must call
#   icp_sal_userStop(...) to free resources.
#   If there are multiple devices present in the system all conf
#   files that describe the devices must have the same address domain
#   sections even if the address domain does not configure any instances
#   on that particular device. So if icp_sal_userStart("xxxxx") is called
#   then user process address domain [xxxxx] needs to be present in all
#   conf files for all devices in the system.
#
# Items configurable by a logical instance are:
# - Name of the logical instance
# - The accelerator associated with this logical
#   instance
# - The execution engine associated with this logical
#   instance (For crypto instances only)
# - The ring bank associated with this logical
```

```
#   instance.
# - The response mode associated wth this logical instance (0
#   for IRQ or 1 for polled).
# - The ring for receiving and the ring for transmitting.
# - The number of concurrent requests supported by a pair of
#   rings on this instance (tx + rx). Note this number affects
#   the amount of memory allocated by the driver. Also
#   Bank<n>InterruptCoalescingNumResponses is only supported for
#   number of concurrent requests equal to 512.
#
# Note: Logical instances may not share the same ring, but
#           may share a ring bank.
#
# The format of the logical instances are:
# - For crypto:
#               Cy<n>Name = "xxxx"
#               Cy<n>AcceleratorNumber = 0
#               Cy<n>ExecutionEngine = 0|1
#               Cy<n>BankNumber = 0-7
#               Cy<n>IsPolled = 0|1
#               Cy<n>NumConcurrentSymRequests = 64|128|256|512|1024|2048|4096
#               Cy<n>NumConcurrentAsymRequests = 64|128|256|512|1024|2048|4096
#               Cy<n>RingAsymTx = 0-14 (Even numbers only)
#               Cy<n>RingAsymRx = 1-15 (Odd numbers only)
#               Cy<n>RingSymTxHi = 0-14 (Even numbers only)
#               Cy<n>RingSymRxHi = 1-15 (Odd numbers only)
#               Cy<n>RingSymTxLo = 0-14 (Even numbers only)
#               Cy<n>RingSymRxLo = 1-15 (Odd numbers only)
#  Note:
#          The value Cy<n>NumConcurrentAsymRequests will do impact to memory
#          consumption greatly. Below is some memory consumption data for
#          the configuration per instance.
#              128: 10M
#              512: 40M
#              1024: 78M
#              4096: 280M
#          By default, 4 kernel instances and 4 user space instances, so if
#          the value is set to be 4096, for pke, the memory consumption is:
#          (4+4)*280=2240M
#
# 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 = 2

# Crypto - Kernel instance #0
Cy0Name = "IPSec0"
Cy0AcceleratorNumber = 0
Cy0ExecutionEngine = 0
Cy0BankNumber = 0
Cy0IsPolled = 0
Cy0NumConcurrentSymRequests = 512
Cy0NumConcurrentAsymRequests = 64
Cy0RingAsymTx = 2
Cy0RingAsymRx = 3
Cy0RingSymTxHi = 4
Cy0RingSymRxHi = 5
Cy0RingSymTxLo = 6
Cy0RingSymRxLo = 7

# Crypto - Kernel instance #1
Cy1Name = "IPSec1"
Cy1AcceleratorNumber = 0
Cy1ExecutionEngine = 1
```



```
Cy1BankNumber = 1
Cy1IsPolled = 0
Cy1NumConcurrentSymRequests = 512
Cy1NumConcurrentAsymRequests = 64
Cy1RingAsymTx = 0
Cy1RingAsymRx = 1
Cy1RingSymTxHi = 2
Cy1RingSymRxHi = 3
Cy1RingSymTxLo = 4
Cy1RingSymRxLo = 5

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

# Crypto - User instance #0
Cy0Name = "SSL0"
Cy0AcceleratorNumber = 0
Cy0ExecutionEngine = 0
Cy0BankNumber = 2
Cy0IsPolled= 1
Cy0NumConcurrentSymRequests = 512
Cy0NumConcurrentAsymRequests = 64
Cy0RingAsymTx =  0
Cy0RingAsymRx =  1
Cy0RingSymTxHi = 2
Cy0RingSymRxHi = 3
Cy0RingSymTxLo = 4
Cy0RingSymRxLo = 5

# Crypto - User instance #1
Cy1Name = "SSL1"
Cy1AcceleratorNumber = 0
Cy1ExecutionEngine = 1
Cy1BankNumber = 3
Cy1IsPolled = 1
Cy1NumConcurrentSymRequests = 512
Cy1NumConcurrentAsymRequests = 64
Cy1RingAsymTx = 0
Cy1RingAsymRx = 1
Cy1RingSymTxHi = 2
Cy1RingSymRxHi = 3
Cy1RingSymTxLo = 4
Cy1RingSymRxLo = 5

##########################################
# User Process Instance Section - Wireless
##########################################
[WIRELESS_INT_0]
NumberCyInstances = 2

# Crypto - User instance #0
Cy0Name = "WIRELESS0"
Cy0AcceleratorNumber = 0
Cy0ExecutionEngine = 0
Cy0BankNumber = 0
Cy0IsPolled = 1
Cy0NumConcurrentSymRequests = 512
Cy0NumConcurrentAsymRequests = 64
Cy0RingAsymTx = 2
Cy0RingAsymRx = 3
Cy0RingSymTxHi = 4
Cy0RingSymRxHi = 5
Cy0RingSymTxLo = 6
Cy0RingSymRxLo = 7

# Crypto - User instance #1
Cy1Name = "WIRELESS1"
Cy1AcceleratorNumber = 0
```

```
Cy1ExecutionEngine = 1
Cy1BankNumber = 1
Cy1IsPolled = 1
Cy1NumConcurrentSymRequests = 512
Cy1NumConcurrentAsymRequests = 64
Cy1RingAsymTx = 2
Cy1RingAsymRx = 3
Cy1RingSymTxHi = 4
Cy1RingSymRxHi = 5
Cy1RingSymTxLo = 6
Cy1RingSymRxLo = 7
```



Appendix B Glossary

ADF	Acceleration Driver Framework
AHCI	Advanced Host Controller Interface
AP	Application Processor
ASIC	Application Specific Integrated Circuit
DID	Device ID
DMA	Direct Memory Access
DTLS	Datagram Transport Layer Security
DRAM	Dynamic Random Access Memory
DRGB	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
EHCI	Enhanced Host Controller Interface
EVP	Envelope (OpenSSL high-level cryptographic functions)
GbE	Gigabit Ethernet
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
NRBG	Non-deterministic Random Number Generator
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
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TRNG	True Random Number Generator
UART	Universal Asynchronous Receiver/Transmitter
UEFI	Unified Extensible Firmware Interface
USB	Universal Serial Bus
VPN	Virtual Private Network