Intel® QuickAssist Technology (Intel® QAT) Software for Linux*

Getting Started Guide -- Customer Enabling Release

Revision 018

April 2025

Document Number: 336212

Legal Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more on Intel's Performance Index site .

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

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

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

See Intel's Legal Notices and Disclaimers .

Contents

1 Introduction 8
1.1 About This Manual 8
1.2 Additional Information on Software 9
1.2.1 Accessing Additional Content from My Intel® 9
1.2.2 Product Documentation 9
1.3 Related Software and Documentation 10
1.4 Conventions and Terminology 10
1.5 Software Overview 11
1.5.1 Features Implemented 11
1.5.2 List of Files in Release 11
1.5.3 Package Release Structure 11

2 Installing the Operating System 12
2.1 Acquiring CentOS* 7 12
2.2 Configure the BIOS 12
2.3 Installing CentOS* 7 13
2.4 Updating Grub Configuration File 14
2.5 Configuring Linux* 14
2.5.1 Updating yum Configuration Files 14
2.5.1.1 /etc/yum.conf 15
2.6 System Security Considerations 15

3 Building and Installing Software 16
3.1 Unpacking the Software 16
3.2 Installation Overview 16
3.2.1 Installation Commands 17
3.2.2 Starting the Driver 18
3.2.3 Installation Procedure 20
3.2.3.1 Standard Installation Procedure 20
3.2.3.2 XEN Installation Procedure 20
3.2.3.3 Platform Cross-Compilation Procedure 22
3.3 Starting/Stopping the Acceleration Software 26
3.4 Configuration Files 27
3.4.1 Configuration File Overview 27
3.4.2 Configuration Files and Virtualization 28

4 Sample Applications 29
4.1 Intel® QAT Accelerator Sample Application 29
4.1.1 Compiling the Acceleration Sample Code 29
4.1.2 Loading the Sample Code 29
4.1.2.1 The signOfLife Tests 31
4.1.3 Test Results 32
4.2 Intel® QAT API Sample Code 32
4.2.1 Compiling the Acceleration Functional Sample Code 32
4.2.2 Executing the Acceleration Functional Sample Code in User Space 33

5 Installing, Building, and Running Yocto* 34

5.1 Building the Yocto* SDK Image 34
5.2 Creating the Linux* Boot Disk 36
5.2.1 Locating the hddimg 36
5.2.2 Creating the Boot Disk 37
5.3 Additional Information on Software 37
5.3.1 Loading the Sample Code 38
5.4 Upgrading Acceleration Software 38
6 Physical Function to Virtual Function Communications 40
Appendix A Avoiding Kernel Crashes with PCH Intel® SKUs 41
A.1 Recommended Procedures 41
A.1.1 Installing the Operating System 41
A.1.2 Operating System First Boot 42
A.2 Alternative Module Load Blocking Procedures 42
A.2.1 Grub Options 42
A.2.2 Changes to Configuration File 43

Tables

Table 1. Product Documentation and Software 9
Table 2. Terminology 10
Table 3. Package Release Structure 11
Table 4. Compile Flag Options 18
Table 5. Staging Contents 23
Table 6. Sample Code Parameters 30

Revision History

Document Number	Revision Number	Description	Revision Date
336212	018	Updates for Intel® QAT Getting Start Guide – Customer Enabling Release include: • Added Section 3.2.3.3 -- Platform Cross-Compilation Procedure	April 2025
336212	017	Updates for Intel® QAT Getting Start Guide – Customer Enabling Release include: • Note added for "make -j" install commands may cause compile issues on some distro's	November 2024
336212	016	Updates for Intel® QAT Getting Start Guide – Customer Enabling Release include: • Notation added to Section 3.3 -- Starting/Stopping the Acceleration Software for stopping services prior to issuing a shutdown to avoid issues with some OSs	July 2024
336212	015	Updates for Intel® QAT Getting Start Guide – Customer Enabling Release include: Table 4. Compile Flag Options Added additional compile flag options: • --enable-aux • --enable-qat19-only • --enable-icp-asan Section 2.2 -- Configure the BIOS: • Recommendation to enable **OS Native support** if available to avoid certain fail signatures.	May 2024
336212	014	Updates for Intel® QAT Getting Start Guide – Customer Enabling Release include: Section 3.2.1 – Installation Commands • Added required assembler from "yasm" to now requiring "nasm", depending on SDK rev. This is due to EOL support for yasm.	November 2023
336212	013	Updates for Intel® QAT Getting Start Guide – Customer Enabling Release include: • Legal Notices & Disclaimers • Section 1.1 – About This Manual • Table 1 – Product Documentation and Software • Section 3.1 – Unpacking the Software • Table 4 – Compile Flag Options • Section 3.2.1 – Installation Commands • Section 3.2.2 – Starting the Driver • Section 3.2.3 – Installation Procedure	March 2023

Document Number	Revision Number	Description	Revision Date
		• Section 3.2.3.1 – Standard Installation Procedure • Section 3.4.1 – Configuration File Overview • Section 3.4.2 – Configuration Files and Virtualization • Section 4.1.1 – Compiling the Acceleration Sample Code • Section 4.1.2 – Loading the Sample Code • Table 5 – Sample Code Parameters • Section 5.4 – Upgrading Acceleration Software	
336212	012	Updates for Intel® QAT Getting Start Guide – Customer Enabling Release: • Name change, now also supports 1.8 HW Gen lookaside (non-inline) features	December 2022
336212	011	Updated the below sections: • Section 3.3 Package Installation using Yum removed, as it is outdated • Section 3.4: Configuration Files completely revised • Table 4. Compile Flag Options added, please note new flag *--enable-legacy-algorithms*	September 2022
336212	010	Updated the below sections: • Table 1.Product Documentation and Software • Section 3.2.3 Installation Procedure • Section 3.2.3.1 Standard Installation Procedure • Section 3.2.3.2 XEN Installation Procedure	March 2022
336212	009	Updated the below sections: • Section 1.5.3 Package Release Structure • Section 2.4 Updating Grub Configuration File • Section 3.2.3 Installation Procedure • Section 4.2.1 Compiling the Acceleration Functional Sample Code	September 2021
336212	008	• Updated package installation requirements • Clarified hardware support	May 2021
336212	007	Minor edits to Appendix A	July 2020
336212	006	Minor edits.	June 2019
336212	005	Added instructions for installing software using yum.	March 2019
336212	004	Updated configuration details.	December 2018
336212	003	Updated information on build instructions and software dependencies.	June 2018

Document Number	Revision Number	Description	Revision Date
336212	002	Added support for Intel® Xeon® Processor D Family devices.	August 2017
336212	001	Initial public release.	July 2017

§

1 Introduction

1.1 About This Manual

This getting started guide documents the instructions to obtain, build, install and exercise the Intel® QAT Software.

Additionally, this document includes brief instructions on configuring the supported development board.

The software described in this document relates to a platform that includes one or more of the following:

- Intel® Communications Chipset 8925 to 8955 Series (formerly "Coleto Creek")
- Intel® C62x Chipset (formerly "Lewisburg")
- Intel® Atom® P6900 processor product family (formerly "Grand Ridge")
- Intel® Xeon® processor product family (formerly "Granite Rapids-D")
- Intel® QAT Adapter 8960/ Intel® QAT Adapter 8970 (formerly "Lewis Hill")
- Intel® Atom® C3000 processor product family (formerly "Denverton")
- Intel® Xeon® Processor D family (formerly "Broadwell-DE")
- Intel® Xeon® Processor D-2100 (formerly "Skylake-D")
- Intel® Xeon® Processor D1700-series (formerly "Ice Lake D")

In this document, for convenience:

- Software package is used as a generic term for the Intel® QuickAssist Technology Software for Hardware CE Version package.
- Acceleration drivers is used as a generic term for the software that allows the Intel® QuickAssist Software Library APIs to access the Intel® QuickAssist Technology Accelerator(s) integrated in the PCH and SoC.

Note: The software package also works on the Intel® Communications Chipset 8925 to 8955 Series.

Sections specific to all covered products include:

- Section 2.0, "Installing the Operating System"
- Section 3.0, "Building and Installing Software"
- Section 4.0, "Sample Applications"

Sections specific to the Intel® Atom® C3000 Processor Product Family SoC include:

- Section 5.0, "Installing, Building, and Running Yocto*"

1.2 Additional Information on Software

The software release package for Linux* has been validated with CentOS* 7 x86_64. It has also been validated with Yocto* for the Intel® Atom® Processor 3000 SoC.

Collateral can be found on https://developer.intel.com/quickassist

1.2.1 Accessing Additional Content from My Intel®

1. In a web browser, go to My Intel.
2. Enter your login ID in the Login ID box. Check Remember my login ID only if you are not using a shared computer. Click **Submit**.

Note: To acquire a new My Intel® Business Applications & Tools, contact your Intel® Field Sales Representative.

3. Enter your password in the Password box. Click **Submit**.
4. Under the **My Memberships** heading, click on **Developer**.
   a. Search for the Code Name of the appropriate device:
      – For the Intel® C62x Chipset PCH, enter the text **Purley** in the text box next to the Magnifying Glass.
      – For the Intel® Atom® C3000 Processor Product Family SoC, enter the text **Denverton NS**.

1.2.2 Product Documentation

Table 1 lists the documentation supporting this release. All documents can be accessed as described in Section 1.2.1, "Accessing Additional Content from the Intel Business Portal".

Table 1. Product Documentation and Software

Document Title	Document Number
Intel® QuickAssist Technology Software for Linux* CE Release Notes	336211
Intel® QuickAssist Technology Software for Linux* Release Notes (Hardware Version 1.8 for In-line)	613775
Intel® QuickAssist Technology Software for Linux* - CE Programmer's Guide	336210
Intel® QuickAssist Technology Cryptographic API Reference Manual	330685
SoC Yocto* BSP - PV	565774
Intel® QuickAssist Technology API Programmer's Guide	330684
Intel® QuickAssist Technology Data Compression API Reference Manual	330686
Intel® QuickAssist Technology - Performance Optimization Guide	330687
Using Intel® Virtualization Technology (Intel® VT) with Intel® QuickAssist Technology Application Note	330689
Intel® QuickAssist Technology Software for FreeBSD* (Berkeley Software Distribution) - Release Notes	621446

1.3 Related Software and Documentation

Refer to the Development Kit User Guide for your hardware for additional information on the development board including board layout, components, connectors, jumpers, headers, power and environmental requirements, and pre-boot firmware.

Follow the directions in Section 1.2.1, "Accessing Additional Content from the Intel Business Portal" to locate this collateral.

1.4 Conventions and Terminology

The following conventions are used in this manual:

- `Courier font` - code examples, command line entries, API names, parameters, filenames, directory paths, and executables
- **Bold text** - graphical user interface entries and buttons
- *Italic text* – key terms and publication titles

The following terms and acronyms are used in this manual:

Table 2. Terminology

Term	Definition
API	Application Programming Interface
BDF	Bus Device Function
BIOS	Basic Input/Output System
BMSM	Broad Market Switch Mode
BOM	Bill of Materials
BTS	Base Transceiver System
CBC	Cipher Block Chaining
CY	Cryptography
DC	Data Compression
DDK	Driver Development Kit
GRUB	Grand Unified Bootloader
Inline	Intel® QAT 1.8 IPsec inline acceleration
Intel® QAT	Intel® QuickAssist Technology
LTTng	Linux* Trace Toolkit Next Generation
OS	Operating System
PCH	Platform Controller Hub
PCI	Peripheral Component Interconnect
PF	Physical Function

Term	Definition
PKE	Public Key Encryption
RDK	Runtime Development Kit
SKU	Stock Keeping Unit
SoC	System-on-a-Chip
SRIOV	Single Root-I/O Virtualization
VF	Virtual Function

1.5 Software Overview

The software is described in the following topics:

- Section 1.5.1, "Features Implemented"
- Section 1.5.2, "List of Files in Release"
- Section 1.5.3, "Package Release Structure"

1.5.1 Features Implemented

Note: For feature details and limitations, if any, refer to the release notes.

1.5.2 List of Files in Release

A Bill of Materials (BOM) is included as a text file in the software package(s).

1.5.3 Package Release Structure

After unpacking the tar file, the directory should contain the following:

Table 3. Package Release Structure

Files/Directory	Comments
`QAT<version>.tar.gz`	Top-level Intel® QAT package
`./filelist`	List of files in this package
`./LICENSE.GPL`	License file
`./versionfile`	Version file
`./quickassist`	Top-level acceleration software directory

§

2 Installing the Operating System

This section describes the process of obtaining, installing, and configuring the operating system (OS) on the development board.

Although this document describes how to install and configure CentOS* 7, Intel® QuickAssist Technology can work with other Linux* distributions, such as Ubuntu* and Fedora*. Refer to notes in Section 3.2.1 for specific installation commands.

2.1 Acquiring CentOS* 7

Note: CentOS* 7 is based on Red Hat Enterprise Linux* 7.3 (or later) and may also be referred to as CentOS* 7.3. In general, using the latest version of CentOS* 6 or CentOS* 7 is recommended, though it is possible that changes to the Linux* kernel in the most recent versions may break some functionality for releases.

CentOS* 7 is a Linux* distribution built on free and open source software. The software package from Intel® does not include a distribution of CentOS* or any other Linux* variant. The software package includes Linux* device driver source developed by Intel®.

CentOS* 7 x86_64 can be obtained from: https://www.centos.org/download.

Note: This document is written with the CentOS* 7 DVD Install Media in mind. Using any Live Media versions is not recommended.

2.2 Configure the BIOS

Update to the latest stable BIOS for your platform.

If the performance achieved from the acceleration software does not meet the advertised capability, some BIOS changes (or other changes) may be required:

- Set the links to train to the highest possible speed, e.g., PCIe* Gen3 instead of PCIe* Gen2 or Gen1.
- Ensure that the link has trained to the expected width, e.g., x8 or x16.
- Disable some CPU power-saving features.

Performance numbers matching the expected performance may not be achievable for all platform configurations or with the default configuration files. Refer to the Intel® QuickAssist Technology - Performance Optimization Guide (Table 1) for more information on achieving best performance.

In the BIOS setup, set the first boot device to be the DVD-ROM drive and the second boot device to be the drive on which CentOS* 7 is to be installed.

It is recommended that **OS Native AER support** is enabled if available, to avoid certain fail signatures.

2.3 Installing CentOS* 7

Note: If you encounter issues installing the operating system, refer to Appendix A, "Avoiding Kernel Crashes with PCH Intel® SKUs" for a possible resolution.

For complete additional (and non-standard) CentOS* installation instructions, refer to the online installation guide at: https://wiki.centos.org/HowTos

This section contains basic installation instructions. For the purposes of this getting started guide, it is assumed that the installation is from a DVD image.

Note: If the hard drive already has an operating system, some of the following steps may be slightly different.

1. When the development board starts, it should begin booting from the CentOS* 7 installation disc. If not, verify the Boot Order settings described in Section 2.2, "Configure the BIOS".
2. At the welcome prompt, select Test this Media & Install CentOS* 7 and click Enter.
3. Select OK to begin testing the installation media.

Note: It is recommended that the CentOS* 7 installation disc is verified prior to an installation of the OS.

4. After verifying the CentOS* 7 installation disc(s), the graphical portion of the installation is loaded. Click Next to continue the installation process.
5. Update DATE & TIME options if required, including the time zone, network time (if desired).
6. SOFTWARE SELECTION. For the best evaluation experience and to avoid any build issues with the acceleration software package, it is recommended to select "Development and Creative Workstation" as the "Base Environment". Select the following "Add-Ons for Selected Environment" as well: "Additional Development", "Development Tools", and "Platform Development". Also select "Virtualization Hypervisor" if virtualization is required (and supported by the acceleration software package).
7. Select INSTALLATION DESTINATION. If the correct target device is not selected, select it. Click Done.
8. Select NETWORK & HOST NAME. In most cases, changing the Ethernet connection from "OFF" to "ON" is desired. Also set the Host name, if required. Click Done. Ensure DHCP is selected.
9. Once all items on the INSTALLATION SUMMARY are configured, select Begin Installation.
10. Set the root password and create a user. When creating a user, select "Make this user administrator" to enable sudo. Select Done and wait for the installation to complete.
11. When the installation completes, the install DVD should be ejected. Remove the DVD and select Reboot when prompted.

Note: If you encounter issues booting after installing the operating system, refer to Appendix A, "Avoiding Kernel Crashes with PCH Intel® SKUs" for a possible resolution.

When the installation completes, continue with Section 2.4, "Updating Grub Configuration File".

2.4 Updating Grub Configuration File

This section details instructions on updating the Grand Unified Bootloader (grub) configuration file.

Note: Root access is required to make grub changes.

If the acceleration software will be used with a virtualized (SR-IOV) environment update grub to add `intel_iommu=on` to the boot options, using the following procedure:

1. Add "intel_iommu=on" to GRUB_CMDLINE_LINUX in grub file /etc/default/grub:

   ```
   GRUB_CMDLINE_LINUX="resume=/dev/mapper/fedora-swap
   rd.lvm.lv=fedora/root rd.lvm.lv=fedora/swap rhgb quiet
   iomem=relaxed intel_iommu=on"
   ```

2. Find the grub.cfg file using:

   ```
   # find / -name "grub.cfg"
   ```

3. Execute the following based on grub.cfg file location:

   ```
   # grub2-mkconfig -o /<path_to_file>/grub.cfg
   ```

4. Reboot system:

   ```
   # shutdown -r now
   ```

Consult the following document for more information on using the acceleration software in a virtualized environment: Using Intel® Virtualization Technology (Intel® VT) with Intel® QuickAssist Technology Application Note (refer to Section 1.2.2, "Product Documentation").

Note: Using the boot flag intel_iommu=on prevents using the QAT physical function (PF) on the host. To use Intel® QAT on the host with this flag, refer to the virtualization app note cited above for the instructions to use Intel® QAT virtual functions (VFs) on the host. access is required to make grub changes.

2.5 Configuring Linux*

Once the operating system is installed, a few configuration items may need to be completed, such as updating the yum configuration files. This section describes these items.

2.5.1 Updating yum Configuration Files

yum is an application that can be used to perform operating system updates. To use yum in a corporate network, the following change may be required.

2.5.1.1 /etc/yum.conf

If the system needs to connect to the internet through a corporate firewall, yum needs to be updated to use the proxy server. Add a line similar to the following in the /etc/ yum.conf file. The line can be added to the end of the file. Contact your network administrator for details on the proxy server, July 2020.

```
proxy=http://<proxy_server:portnum>
```

where <proxy_server:portnum> is replaced with your server information.

2.6 System Security Considerations

This section contains a high-level list of system security topics. Specific OS/filesystem topics are outside of the scope of this document. For more information, refer to the programmer guide for your platform, specifically the Secure Architecture Considerations section.

Securing your operating system is critical. Consider the following items:

Note: This is not an exhaustive list.

- Employing effective security policies and tools; for instance, SELinux* is configured correctly and is active.
- Running and configuring the firewall(s).
- Preventing privilege escalation at boot (including recovery mode); for instance, set a grub password. Additional details are described below.
- Removing unnecessary software packages.
- Patching software in a timely manner.
- Monitoring the system and the network.
- Configuring and disabling remote access, as appropriate.
- Disabling network boot.
- Requiring secure passwords.
- Encrypting files, up to full-disk encryption.
- Ensuring physical security of the system and the network.
- Using mlock to prevent swapping sensitive variables from RAM to disk.
- Zeroing out sensitive variables in RAM.

§

3 Building and Installing Software

This chapter provides details on building and installing the software.

3.1 Unpacking the Software

The software package comes in the form of a tarball. Refer to Section 1.2.1, "Accessing Additional Content from the Intel Business Portal" for the software location.

The software package can also be installed under Linux* using yum. Refer to Section 3.3, "Package Installation Using yum" for additional information.

The instructions in this document assume that you have super user privileges:

```
# sudo su
<enter password for root>
```

1. Create a working directory for the software. This directory can be user defined, but for the purposes of this document, a recommendation is provided:

   ```
   # mkdir /QAT && cd /QAT
   ```

Note: In this document, the working directory is assumed to be /QAT. This directory is the `ICP_ROOT`.

2. Transfer the tarball to the development board using any preferred method, for example USB memory stick, CDROM, or network transfer in the /QAT directory. Unpack the tarball using the following command:

   ```
   # tar -zxof <QAT tarball name>
   ```

3. Restricting access to the files is recommended:

   ```
   # chmod -R o-rwx *
   ```

Result: The package is unpacked and the installation script and other items are created in the /QAT directory. Refer to Section 1.5.3, "Package Release Structure".

3.2 Installation Overview

The installation procedure handles a number of tasks that would otherwise have to be done manually, including the following:

- Create the shared object (.so) files by building the source code.
- Copy the shared object (.so) files to the right directory (e.g., /lib or /lib64).
- Build adf_ctl and copy it to the right directories ($ICP_ROOT/build and /usr/ sbin).
- Copy the config files to /etc.
- Copy the firmware files to /lib/firmware.

- Copy the modules to the appropriate kernel source directory for loading by qat_service.
- Start the qat_service, which inserts the appropriate modules as required and runs adf_ctl to bring up the devices.
- Set up the qat_service to run on future boots (copy to /etc/init.d, run chkconfig to add the service).

On recent Linux* kernels, there is an upstreamed version of the Intel® QuickAssist Technology driver, and it will interfere with the loading of the driver included with the software package assumed in this document. The qat_service accounts for this by removing the upstreamed kernel modules, but if qat_service is not used, errors may be displayed when trying to load the driver.

3.2.1 Installation Commands

Note: If the OS was not installed with the correct software packages (refer to Section 2.3, "Installing CentOS* 7"), build error messages appear during the acceleration install. If CentOS* was not installed correctly, reinstall the OS and select the correct SOFTWARE SELECTION option as described in Section 2.3, "Installing CentOS* 7", or run the following commands:

```
yum -y groupinstall "Development Tools"
yum -y install pciutils
yum -y install libudev-devel
yum -y install kernel-devel-$(uname -r)
yum -y install gcc
yum -y install openssl-devel
```

Note: If you are installing Ubuntu*, run these commands instead:

```
apt-get update
apt-get install pciutils-dev
apt-get install g++
apt-get install pkg-config
apt-get install libssl-dev
```

Note: If you are installing a recent distribution of Fedora*, run these commands instead:

```
dnf groupinstall "Development Tools"
dnf install gcc-c++
dnf install systemd-devel
dnf install kernel-devel-`uname -r`
dnf install elfutils-devel
dnf install openssl-devel
```

Note: Installing nasm (v2.14+) or yasm, as well as the Netlink Protocol Library Suite, may be required depending on your SDK package. For CentOS*, enabling the Power Tools repo may also be required. For instance, to install yasm, nasm and libnl on CentOS* 7 or 8, use "yum install epel-release; yum install yasm; yum install nasm; yum install libnl3-devel".

3.2.2 Starting the Driver

To start the driver, use the qat_service, or rmmod the upstreamed modules (qat_*, intel_qat) and insert the modules built with the software package assumed in this document before starting the driver.

The acceleration software package supports the standard Linux* software installation process.

```
# ./configure [OPTION]... [VAR=VALUE]
# make -j install
```

Note: Using the "-j" simultaneous job option may cause compile issues on some distro's e.g. Fedora 40.

Run the following command to see the list of available options:

```
# ./configure --help
```

Note: If you are installing an Intel® C629 Series Chipset or any Intel® device that does not support Public Key Encryption (PKE), enter the following configuration option:

```
# ./configure --enable-icp-dc-sym-only
```

This option enables the data compression and symmetric code services and disables cryptographic services.

Note: To enable the Intel® QuickAssist API in Kernel space, enter the following configuration option:

`# ./configure --enable-kapi` (not supported for all 1.8 HW devices i.e. c4xxx)

Note: Current implementation of Intel® QuickAssist API for Kernel space only supports Symmetric cryptography and Data compression services.

Table 4. Compile Flag Options

Compile Flag	Description
`--enable-icp-debug`	Enables debugging.
`--enable-qat-uio`	Enables Userspace I/O.
`--disable-option-checking`	Ignore unrecognized --enable/--with options.
`--disable-FEATURE`	Do not include FEATURE (same as --enable-FEATURE=no).
`--enable-FEATURE[=ARG]`	Include FEATURE [ARG=yes].
`--enable-silent-rules`	Less verbose build output (undo: "make V=1").
`--disable-silent-rules`	Verbose build output (undo: "make V=0").
`--disable-param-check`	Disables parameters checking in the top-level APIs (used for performance optimization).
`--disable-dc-dyn`	Disables Dynamic compression support.
`--disable-stats`	Disables statistic collection (used for performance optimization).

Compile Flag	Description
`--disable-dc-strict-mode`	Disables strict mode for data compression.
`--enable-icp-log-syslog`	Enables debugging messages to be outputted to the system log instead of standard output.
`--enable-icp-sriov`	Enables Single-root I/O (SR-IOV) Virtualization in the Intel®® QAT driver (available options: host, guest).
`--enable-icp-trace`	Enables tracing for the Cryptography API.
`--enable-icp-dc-only`	Enables driver supports only compression service (can optimize the size of build objects).
`--enable-icp-dc-sym-only`	Enables drivers to support Data Compression and Symmetric Crypto services only.
`--enable-icp-dc-return counters-on-error`	Enables updates of consumed/produced results in case of error during compression or decompression operations.
`--enable-icp-hb-fail-sim`	Enable Heartbeat Failure Simulation.
`--enable-inline`	Enables the Intel® QAT 1.8 IPsec inline acceleration feature.
`--enable-bmsm`	Enables the Intel® QAT 1.8 IPsec BMSM feature.
`--enable-lttng`	Enables the Intel® QAT 1.8 LTTng feature for inline in BTS mode
`--enable-qat-coexistence`	Enables legacy and upstream driver coexistence.
`--enable-qat-lkcf`	Enables Intel® QAT registration with Linux* Kernel Crypto Framework.
`--enable-dc-error simulation`	Enables Data Compression Error Simulation.
`--enable-kapi`	Enables Intel® QuickAssist API in Kernel space (not supported for all 1.8 HW devices i.e. c4xxx).
`--disable-stats`	Disables statistics collection.
`--enable-legacy algorithms`	Enables legacy algorithms. See Intel® QAT Programmer's Guide *3.22 Access to Legacy Algorithms*
`--enable-icp-qat-dbg`	Enable QAT Debuggability.
`--enable-qat-xen`	Enables building QAT for Xen system.
`--enable-icp-without-thread`	Removes mutex and spin locks for single thread applications.
`--enable-icp-thread-specific-usdm`	USDM allocates and handles memory specific to threads (for multi-thread apps, allocated memory information will be maintained separately for each thread).
`--enable-128k-slab`	Enables 128k slab allocator in USDM. It could improve performance and reduce memory consumption for the large number of threads when thread specific memory allocator is enabled.
`--enable-aux`	Enables AUX feature by default (required for Media offload) – applicable on QAT2.2 SDK/HW
`--enable-qat19-only`	Enables QAT1.9 HW only – applicable on QAT2.2 SDK/HW
`--enable-icp-asan`	Enables address sanitization – available on CE v4.25+

Compile Flag	Description
	Note: This feature may have performance impacts and should only be used for debug purposes.

3.2.3 Installation Procedure

When installing acceleration software on a system that had a previous or modified version of the acceleration software installed, it is strongly recommended to uninstall the previous acceleration software first, using the following command in the acceleration software package:

```
# make uninstall && make clean && make distclean
```

3.2.3.1 Standard Installation Procedure

1. Open a Terminal Window and switch to superuser:

```
# sudo su <enter root password>
# cd /QAT
```

2. Enter the following commands to build and install the acceleration software and sample code using default options:

```
# ./configure
# make -j install
# make -j samples-install
```

Note: Using the "-j" simultaneous job option may cause compile issues on some distro's e.g. Fedora 40.

3.2.3.2 XEN Installation Procedure

1. Open a Terminal Window on DDK setup and switch to superuser:

```
# su <enter root password>
# cd /QAT
```

2. Enter the following commands to build the driver:

```
# ./configure --enable-qat-xen
# make
```

3. Copy output "build" directory containing firmware and kernel modules to the XEN host server – typically copied to the XEN /root directory:

```
# scp -r build <xen_ip_address>:/<xen_path>/
```

4. Log into the XEN server and copy over firmware modules:

```
# ssh <user>@<xen_ip_address>
# su <enter root password>
# cd <xen_path>/build
# cp -a *.bin /lib/firmware
```

5. Remove any existing drivers and load new XEN required drivers:

```
# modprobe uio
# modprobe authenc
# rmmod intel_qat
# insmod intel_qat.ko
```

For Intel® C62X CHIPSET:

```
# rmmod qat_c62x
# insmod qat_c62x.ko
# cp <xen_path>/build/c6xx_dev0.conf.xen /etc/c6xx_dev0.conf
```

For Intel® Communication Chipset 8925 T0 8955 Series:

```
# rmmod qat_dh895xcc
# insmod qat_dh895xcc.ko
# cp <xen_path>/build/dh895xcc_dev0.conf.xen /etc/dh895xcc_dev0.conf
```

Note: XEN supports only crypto services, so ensure that ServiceProfile is set as "CRYPTO", ServicesEnabled is set to "cy" and NumberDcInstances is set to "0" in device config files for both PF and VF devices (including the guest operating system):

6. Emulate VFs:

```
# ./adf_ctl restart
# echo 1 > /sys/bus/pci/devices/<PF_DEVICE>/sriov_numvfs
```

7. At this point, the QAT VFs can be mapped to FreeBSD guest operating systems.

Note: For selecting the correct QAT driver for use in the guest operating systems using FreeBSD, consult the Intel® QuickAssist Technology Software for Free Berkeley Software Distribution (refer to Section 1.2.2, "Product Documentation").

Note: When configuring the QAT driver in the guest operation systems, ensure "enable-icp-sriov=guest" is selected. For further details using guest operating systems, consult the Intel® Virtualization Technology (Intel® VT) with Intel® QuickAssist Technology Application Note (refer to Section 1.2.2, "Product Documentation").

Note: To build 64-bit kernel components and 32-bit user-space components, run the make install command with the following build parameter:

```
# make ICP_ARCH_USER=i686 install
```

Note: After building/installing the acceleration software, secure the build output files by either deleting them or setting permissions according to your needs.

Note: The messages "Can't read private key" can be safely ignored. These are generated because the modules are not signed by the private key of OS distribution.

8. Once the Acceleration Software is installed, it is recommended to verify that the acceleration software kernel object is loaded and ready to use:

```
# lsmod | grep qa
```

Depending on the specific hardware present, this command returns something similar to the following:

```
# lsmod | grep qa

qat_c62x          18061 0
intel_qat         214941 2 usdm_drv,qat_c62x
uio               19338 1 intel_qat
```

Not all modules are required, depending on the specific hardware present. If the acceleration software is not installed, all of these modules are typically not present.

Note: If the ./configure command would have been run with option --enable-kapi, the QuickAssist API Kernel module would be loaded as part of the installation and the console command # lsmod | grep qa would return something similar to:

```
# lsmod | grep qa
qat_api 561152    0
qat_c62x          20480   0
intel_qat         225280  3 qat_c62x,qat_api,usdm_drv
authenc 16384     1 intel_qat
dh_generic        16384   1 intel_qat
uio               20480   1 intel_qat
```

The Kernel module qat_api exports the Intel® QuickAssist APIs as symbols allowing a Kernel space application to use them.

Applications need to make use of the static library (libqat.a) or the shared object (libqat_s.so). If the installation procedure described in this chapter is not used, the shared object needs to be copied manually, or other steps (e.g., setting LD_LIBRARY_PATH) need to be taken to link to this file.

Check /var/log/messages or dmesg to make sure that the acceleration service started. Warning messages related to Invalid core affinity can be addressed by modifying the configuration files so that no core numbers are referenced beyond the core count of the system. Refer to Section 3.5, "Configuration Files" for more detail.

Once the installation/building is complete, proceed to Section 4.0, "Sample Applications" to execute applications that exercise the software.

3.2.3.3 Platform Cross-Compilation Procedure

The following describes procedures for using a build server to compile the Intel® QuickAssist SDK driver package, followed by deploying the required modules onto a target server and subsequently enabling the Intel® QuickAssist device.

Note: Applies to Intel® QuickAssist CE driver package v4.28 or later

Prerequisites:

- Copy the kernel headers from the target host to the build host – these are typically located in /lib/modules/$(uname -r)/build.

Steps performed on build host:

1. Create a working directory for the software. This directory can be user defined, but for the purposes of this document a recommendation is provided:

```
# mkdir /QAT && cd /QAT
```

Note: In this document, the working directory is assumed to be `/QAT`. This directory is the `ICP_ROOT`.

2. Transfer the tarball to the development board using any preferred method, for example, the USB memory stick, CDROM, or network transfer in the /QAT directory.

3. Unpack the tarball using the following command:

```
# tar -zxof <QAT tarball name>
```

4. Restricting access to the files is recommended:

```
# chmod -R o-rwx *
```

5. Prepare the environment:

- Define the ICP_ROOT QAT working directory e.g.:

  # export ICP_ROOT=/QAT

- Define the location of the target kernel headers e.g.:

  # export TARGET_HEADERS=/home/user/target_headers_dir

- Specify the destination for the redirected QAT installation e.g.:

  # export QAT_STAGING=/home/user/qat_staging

6. Configure, build and install QAT:

- Run the configuration script providing the KERNEL_SOURCE_ROOT:

  # ./configure KERNEL_SOURCE_ROOT=$TARGET_HEADERS

- Install the QAT package with installation redirection. Ensure both DESTDIR and INSTALL_MOD_PATH are specified:

  # make install DESTDIR=$QAT_STAGING INSTALL_MOD_PATH=$QAT_STAGING

- Optionally, install the QAT sample code:

  # make samples-install DESTDIR=$QAT_STAGING INSTALL_MOD_PATH=$QAT_STAGING

7. Verify staging directory contents using table below:

Table 5. Staging Contents

Location	Description
`$QAT_STAGING/lib/firmware/qat*.bin`	QAT Firmware for all supported 1.x devices
`$QAT_STAGING/lib/modules/<kernel_version>/updates/drivers/crypto/qat/qat_*/*.ko`	QAT kernel modules for all supported 1.x devices

Location	Description
$QAT_STAGING/lib/modules/<kernel_version>/kernel/drivers/qat_api.ko	QAT kernel API driver
$QAT_STAGING/lib/modules/<kernel_version>/kernel/drivers/usdm_drv.ko	QAT memory driver
$QAT_STAGING/etc/init.d/qat_service $QAT_STAGING/etc/init.d/qat_service_vfs [optional]	QAT service script to manage the PF/VF
$QAT_STAGING/etc/default/qat	Configuration file for the QAT service
$QAT_STAGING/usr/local/lib/libusdm_drv_s.so	User-space shared library for QAT API
$QAT_STAGING/etc/ld.so.conf.d/qat.conf	Configuration file for ldconfig
$QAT_STAGING/etc/udev/rules.d/00-qat.rules	QAT rules file for Udev
$QAT_STAGING/usr/local/bin/adf_ctl	Control application for QAT HW
$QAT_STAGING/usr/local/bin/cpa_sample_code [optional]	User-space sample code binary
$QAT_STAGING/usr/local/bin/cpa_sample_code.ko	Kernel-space sample code
$QAT_STAGING/lib/firmware/calgary [optional] $QAT_STAGING/lib/firmware/canterbury [optional] $QAT_STAGING/lib/firmware/calgary32 [optional]	Input text files used by QAT sample code

Steps performed on target host:

1. Prepare the environment:

- Copy the entire staging directory from the build to the target host e.g. $QAT_STAGING directory
- Blacklist QAT in-tree modules.

2. Install QAT software – example provided uses qat_c4xxx device:

Backup existing FW before copying the new files

[optional] sudo mv /lib/firmware/qat_c4xxx.bin /lib/firmware/qat_fw_backup/

[optional] sudo mv /lib/firmware/qat_c4xxx_mmp.bin /lib/firmware/qat_fw_backup/

Note: Original file permissions should be preserved throughout the entire deployment process.

sudo cp $QAT_STAGING/lib/firmware/qat_c4xxx.bin /lib/firmware/

sudo cp $QAT_STAGING/lib/firmware/qat_c4xxx_mmp.bin /lib/firmware/

sudo cp $QAT_STAGING/lib/modules/$(uname -r)/updates/drivers/crypto/qat/qat_common/intel_qat.ko /lib/modules/$(uname -r)/updates/drivers/crypto/qat/qat_common/

sudo cp $QAT_STAGING/lib/modules/$(uname -r)/updates/drivers/crypto/qat/qat_c4xxx/qat_c4xxx.ko /lib/modules/$(uname -r)/updates/drivers/crypto/qat/qat_c4xxx/

sudo cp $QAT_STAGING/lib/modules/$(uname -r)/kernel/drivers/qat_api.ko /lib/modules/$(uname -r)/kernel/drivers/

sudo cp $QAT_STAGING/lib/modules/$(uname -r)/kernel/drivers/usdm_drv.ko /lib/modules/$(uname -r)/kernel/drivers/

sudo cp $QAT_STAGING/etc/init.d/qat_service /etc/init.d/
sudo cp $QAT_STAGING/etc/default/qat /etc/default/

sudo cp $QAT_STAGING/usr/local/lib/libqat_s.so /usr/local/lib/

sudo cp $QAT_STAGING/usr/local/lib/libusdm_drv_s.so /usr/local/lib/

sudo cp $QAT_STAGING/etc/ld.so.conf.d/qat.conf /etc/ld.so.conf.d/

sudo cp $QAT_STAGING/etc/udev/rules.d/00-qat.rules /etc/udev/rules.d/

sudo cp $QAT_STAGING/usr/local/bin/adf_ctl /usr/local/bin/

Install QAT sample code and dependencies

[optional] sudo cp $QAT_STAGING/usr/local/bin/cpa_sample_code /usr/local/bin/

[optional] sudo cp $QAT_STAGING/usr/local/bin/cpa_sample_code.ko /usr/local/bin/

[optional] sudo cp $QAT_STAGING/lib/firmware/calgary /lib/firmware/

[optional] sudo cp $QAT_STAGING/lib/firmware/calgary32 /lib/firmware/

[optional] sudo cp $QAT_STAGING/lib/firmware/canterbury /lib/firmware/

3. Configure the target host.

Note: The build host does not have information about the Intel® QuickAssist hardware available on target host or the operating system it is running. Users must configure the OS and load the appropriate QAT configuration files. Default configuration files for all QAT 1.x devices can be found in the unpacked tarball at $ICP_ROOT/quickassist/utilities/adf_ctl/conf_files/.

4. Minimal OS configuration example:

- Generate a list of module dependencies and map files for the Linux Kernel:

  # depmod -a

- Configure dynamic linker run-time bindings:

  # ldconfig

- Register and start the QAT service (an example for systemd based hosts):
  - Manually copy the QAT unit file from the build host to the target host. The unit file is located in the unpacked tarball at $ICP_ROOT/*quickassist/build_system/build_files/qat.service* and should be placed in */usr/lib/systemd/system/.*
  - Enable and start the service:

    # systemctl enable qat

    # systemctl daemon-reload

    # systemctl start qat

Note: Additional configurations may be required for specific use cases, as described in other sections of this document.

Uninstallation:

Build Host:

To uninstall QAT files from the staging directory without affecting build host's system directories, use:

make uninstall DESTDIR=$QAT_STAGING INSTALL_MOD_PATH=$QAT_STAGING

Target Host:

Uninstallation can be done manually by reversing the changes done during the manual installation process.

Alternatively, a fresh QAT package can be copied to the target host and the following command can be used to remove the QAT software automatically:

./configure && *make uninstall*

3.3 Starting/Stopping the Acceleration Software

When the acceleration software is installed, a script file titled qat_service is installed in the /etc/init.d directory.

The script file can be used to start and stop the acceleration software. To start the software, issue the following command:

```
# service qat_service start
```

Note: If the service qat_service start command fails, verify the following:

- The software is installed.
- Acceleration software is already running.
- For the Intel® C62x Chipset or Intel® Xeon® Processor D Family SoC, verify the device is enumerated properly using the lspci command:

  ```
  # lspci -d 8086:37c8
  ```

- For the Intel® Atom® C3000 Processor SoC, verify the device is enumerated properly using the lspci command:

  ```
  # lspci -d 8086:19e2
  ```

- For the Intel® Xeon® Processor D Family SoC, verify the device is enumerated properly using the lspci command:

  ```
  # lspci -d 8086:6f54
  ```

To stop the software, issue the following command:

```
# service qat_service stop
```

To stop the software and remove the kernel driver, issue the following command:

```
# service qat_service shutdown
```

Note: Services should be stopped using `# service qat_service stop` prior to shutdown to avoid issues on some OSs e.g. Ubuntu.

When the acceleration software is installed, it is set to load automatically when the operating system loads.

3.4 Configuration Files

Note: This section on configuration files is only a brief overview. For more details, refer to the Programmer's Guide for your platform for complete information on the configuration files. Usually this is in Chapter 4: Configuration File.

3.4.1 Configuration File Overview

- The software package includes multiple types of platform-specific *default* configuration files. Depending on your installation options and SKU, a valid configuration file is copied to the /etc directory. If your system has more than one type of hardware device or SKU, verify that the correct configuration files were copied.
- The files are processed when the system boots. **If changes are made to the configuration file, the Acceleration software must be stopped and restarted for the changes to take effect.** Refer to Section 3.2.3, "Installation Procedure" on for detailed instructions.
- There is a single configuration file for each QAT endpoint, including virtual functions (VFs, see below).
- Configuration files are placed in the `/etc/` directory and terminate with a **`.conf`** file extension. Examples:
  - /etc/dh895xcc_dev0.conf (first configuration file for Intel® Communications Chipset 8925 to 8955 Series devices)
  - /etc/c6xx_dev0.conf (first configuration file for Intel® C62x Chipset or Intel® Xeon® Processor D Family SoC)
  - /etc/c3xxx_dev0.conf (first configuration file for Intel® Atom® C3000 Processor SoC)

Users may need to confirm the main sections in the configuration file(s), including:

- [GENERAL]
- # Kernel Instances Section
- # User Process Instance Section (**[SSL]** by default, but may be renamed)

In particular:

- Ensure **`ServicesEnabled`** is appropriate for the device and application (generally cryptography and/or compression)
- **ServicesProfile** may disable/enable features for your device and use case
- Confirm the following for *both* **[KERNEL]** and **[SSL]** sections:
  - NumberCyInstances
  - NumberDcInstances

This is only an overview on configuration files. For detailed information, refer to the Programmer's Guide for your platform. Usually this is Chapter 4: Acceleration Driver Configuration File.

3.4.2 Configuration Files and Virtualization

Virtual Function (VF) configuration filenames are usually off-by-one from the qat_dev#, as numbering begins after all PFs. With one PF device as `qat_dev0`, for example, VF `qat_dev_5` will use configuration file `c4xxx_vf_dev6.conf`:

- /etc/c4xxx_**vf_dev0**.conf (VF qat_qat_dev**1**)
- /etc/c4xxx_**vf_dev6.**conf (VF qat_qat_dev**5**)

Tips:

- Ensure none of the VFs are requesting services i.e., cy or dc, which are not enabled in the PF config file.

For more information on Virtualization, refer to our Virtualization Guide, Document # 330689, at https://cdrdv2.intel.com/v1/dl/getContent/709210.

§

4 Sample Applications

This chapter describes the sample code that can be executed on the target platform along with instructions on their usage.

4.1 Intel® QAT Accelerator Sample Application

The software package contains a set of sample tests that exercise acceleration functionality. This section describes the steps required to build and execute the sample tests.

The sample application is provided for the user space.

4.1.1 Compiling the Acceleration Sample Code

Note: These instructions assume the software package was untarred in the /QAT directory and the kernel source files were placed in the directory specified in this guide.

1. Open a Terminal Window and switch to superuser:

```
# sudo su
<enter root password>
```

Note: For details on running user space applications as non-root user, refer to the "Running Applications as Non-Root User" section in the applicable programmer guide (refer to Section 1.2.2, "Product Documentation").

2. Switch to the /QAT directory and compile the installation samples:

```
# cd /QAT
# make -j samples-install
```

Note: Using the "-j" simultaneous job option may cause compile issues on some distro's e.g. Fedora 40.

This compiles the acceleration sample code for user space; refer to section 4.2 for more details. It also compiles the memory mapping driver used with the user space application.

Proceed to "Running the chained hash and compression test code requires:" for instructions on executing the tests.

4.1.2 Loading the Sample Code

Note: In user space, before launching the cpa_sample_code application, the environmental variable LD_LIBRARY_PATH may need to be set to the path where libqat_s.so is located. This may be /usr/local/lib or /QAT/build.

The acceleration kernel module must be installed, and the software must be started before attempting to execute the sample code. The module can be verified by running the following

commands:

```
# lsmod | grep "qa"
# service qat_service status
```

The typical output is similar to the following:

```
# service qat_service status Checking status of all devices.
There is 3 QAT acceleration device(s) in the system:
qat_dev0 - type: c6xx, inst_id: 0, bsf: 88:00.0, #accel: 5 #engines: 10
state: up qat_dev1 - type: c6xx, inst_id: 1, bsf: 8a:00.0, #accel: 5
#engines: 10 state: up qat_dev2 - type: c6xx, inst_id: 2, bsf: 8c:00.0,
#accel: 5 #engines: 10 state: up
```

Note: If the module is not returned from the first command, refer to Section 3.2.3, "Installation Procedure" for additional information on starting the Acceleration software.

In user space, the sample code is executed with the command:

```
# ./build/ cpa_sample_code
```

In Kernel space the sample code is executed with the command:

```
insmod ./build/cpa_sample_code.ko
```

Note: The output of the cpa_sample_code application is available in `dmesg`.

Should the user decide to run the sample code more than once, unload the `cpa_sample_code` module as follows before rerunning it:

```
# rmmod cpa_sample_code
```

The application allows the run-time parameters listed below.

Table 6. Sample Code Parameters

Parameter	Supported in User Space	Supported in Kernel Space	Description
`cyNumBuffers=w`	Yes	Yes	The number of buffers submitted for each iteration. (default=20)
`cySymLoops=x`	Yes	Yes	The number of iterations of all symmetric code tests. (default=5000)
`cyAsymLoops=y`	Yes	No	The number of iterations of all asymmetric code tests. (default=5000)
`runTests=1`	Yes	Yes	Run symmetric code tests.
`runTests=2`	Yes	No	Run the RSA test code.
`runTests=4`	Yes	No	Run the DSA test code.
`runTests=8`	Yes	No	Run the ECDSA test code.
`runTests=16`	Yes	No	Run the Diffie-Hellman code tests.

Parameter	Supported in User Space	Supported in Kernel Space	Description
`runTests=32`	Yes	Yes	Run the compression code tests.
`runTests=63`	Yes	Yes	Run all tests except the chained hash and compression tests. (default)
`runTests=128`	Yes	Yes	Run chained hash and compression test code.
`runStateful=1`	Yes	Yes	Enable stateful compression tests. Applies when compression code tests are run.
`signOfLife=1`	Yes	Yes	Indicates shorter test run that verifies the acceleration software is working. This parameter executes a subset of sample tests. Details are included in "Running the chained hash and compression test code requires:". (default=0)
`getLatency=1`	Yes	No	Measures the processing time for the request being processed. Requires NumberCyInstances=1 and NumberDcInstances=1 to be configured in [SSL] section of the driver configuration file.
`getOffloadCost=1`	Yes	No	Measures the average number of cycles spent for single- request offloading. Requires NumberCyInstances=1 and NumberDcInstances=1 to be configured in [SSL] section of the driver configuration file.

Note: Running the chained hash and compression test code requires the following:

- Updating the configuration file to set `ServicesProfile = COMPRESSION`
- Restarting qat_service
- Possibly setting ServicesEnabled to `dc` or `dc;sym`.

4.1.2.1 The signOfLife Tests

The `signOfLife` parameter is used to specify that a subset of the sample tests is executed with smaller iteration counts. The `signOfLife` test provides a quick test to verify that the acceleration software and hardware are set up correctly.

Note: If the signOfLife parameter is not specified, the full run of tests can take a significant amount of time to complete.

4.1.2.1.1 User Space

After building the sample code with the installation script, the user space application is located at:

```
$ICP_ROOT/build
```

Then, run the following commands:

```
# cd $ICP_ROOT/build/
# export LD_LIBRARY_PATH=`pwd`
# ./cpa_sample_code signOfLife=1
```

4.1.2.1.2 Kernel Space

After building the sample code with the installation script, the kernel space application is located at:

```
$ICP_ROOT/build
```

Then run the following commands:

```
# cd $ICP_ROOT/build/
# insmod ./cpa_sample_code.ko signOfLife=1
```

On a second terminal window, run `dmesg` to view the output of the `cpa_sample_code` Kernel application.

4.1.3 Test Results

When running the application, the results are printed to the terminal window in which the application is launched.

Here is an example of the log messages created during the test:

```
Algorithm Chaining - AES256-CBC HMAC-SHA512 Number of threads  2
Total Submissions 20
Total Responses   20
Packet Size       512
```

A similar pattern is repeated for each of the tests.

4.2 Intel® QAT API Sample Code

The software package contains sample code that demonstrates how to use the Intel® QAT APIs and build the structures required for various use cases.

For more details, refer to the Intel® QuickAssist Technology API Programmer's Guide (refer to the listing in Table 1).

4.2.1 Compiling the Acceleration Functional Sample Code

The acceleration functional sample code can be compiled manually.

Note: These instructions assume the software package has been untarred to the /QAT directory and the kernel source files were placed in the directory specified in this guide.

1. The following environment variable must be set to build the modules:

   ```
   # export ICP_ROOT=<QATdir>
   ```

   where <QATdir> is /QAT or the directory where the package was untarred.
2. Compile for the user space using the following commands:

   ```
   # cd
   $ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/function
   al
   # make all
   ```

Result: The generated Linux* kernel objects and sample applications are located at:
`$ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/functional/b
uild`

4.2.2 Executing the Acceleration Functional Sample Code in User Space

To execute the acceleration functional sample code in user space, use the following commands:

```
# cd
$ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/function
al/ build
”./hash_file_sample
```

Note: The hash_file_sample is one of the functional user space applications. Launch the other user space applications similarly.

§

5 Installing, Building, and Running Yocto*

The Yocto Project* is an open-source collaboration project focused on embedded Linux*. Yocto* includes a set of tools to build a custom Linux* distribution. The process to create your custom Linux* distribution using Yocto* involves creating your own image on a software development workstation. The steps in Section 5.1, "Building the Yocto* SDK Image" should be done on a software development workstation, not the Harcuvar CRB.

The steps to build and copy the image on Ubuntu* 14.04 are included here. If using a different Linux* distribution, consult the Yocto Project* website -https://www.yoctoproject.org) for more information and documentation, including:

- Yocto* Quick Start Guide: http://www.yoctoproject.org/docs/latest/yocto-project-qs/yocto-project-qs.html
- Git repository: http://git.yoctoproject.org/cgit/cgit.cgi/meta-intel/

Note: The pre-built image has a Time-Limited-Kernel (TLK), which means that the image is restricted to a 10-day uptime and the image will be auto-rebooted after that time. TLK is added to encourage end-users to build their own image for production.

5.1 Building the Yocto* SDK Image

Follow the instructions below to create the Yocto* SDK image. A pre-built image Yocto* SDK image is provided in the Electronic Design Kit (document number 565762). (refer to listing in Table 1, "Product Documentation and Software"). If using the pre-built image, proceed to Section 5.2, "Creating the Linux* Boot Disk" for instructions on creating OS boot image.

Note: If you are upgrading the acceleration drivers, proceed to Section 5.4, "Upgrading Acceleration Software".

Prerequisites: The build process using sato consumes about 100 GB of disk space. Therefore, at least 200 GB of free disk space is recommended.

Note: If script build errors appear to be syntax errors, it is likely that the script is being passed to the wrong shell. Many Yocto* scripts call /bin/sh, which may be symbolically linked to /bin/dash. To resolve the problem, remove /bin/sh ("sudo rm /bin/ sh") and link to bash instead ("sudo ln -s /bin/bash /bin/sh").

1. Install Ubuntu* 14.04 (64-bit).

Note: Always execute the following instructions as a non-root user.

2. If required, add the following proxy settings for your network environment to /etc/environment:

```
https_proxy='https://<proxy_server>:<proxy_port>/'
http_proxy='http://<proxy_server>:<proxy_port>/'
ftp_proxy='http://<proxy_server>:<proxy_port>/'
GIT_PROXY_COMMAND=/usr/bin/git_proxy_command
```

Configure ssh configuration file for proxy settings. Add following lines to `~/.ssh/config`:

```
host *
ProxyCommand connect-proxy -s %h %p
```

Create the /usr/bin/git_proxy_command file and add the following lines:

```
#!/bin/sh
connect-proxy -s $@
```

Change the git_proxy_command file to be executable:

```
sudo chmod +x /usr/bin/git_proxy_command
```

Log out of the current user account and login, to allow the environment changes to take effect.

3. Update apt-get:

```
# sudo apt-get -y update
```

4. Install the required software components:

```
# sudo apt-get -y install gawk wget git-core diffstat unzip texinfo \
build-essential chrpath libsdl1.2-dev xterm socat connect-proxy
```

Note: If the apt-get command fails completely, try it again, since the specific mirror selected may not have transferred the files correctly.

5. Download the BSP (document number 565774) and copy this file to the `/home/<userid>` directory on the build system. Then change that directory.
6. Decompress and extract the archived package. For example:

```
<package_name> = harcuvar_PV
# tar xzvf <package_name>.tar.gz
```

The <package_name> package consists of setup folder and setup scripts. The setup script has test logic to check if your build system can access the Internet.

The setup folder consists of BSP-related patches inside `setup/patchset/bsp`, `setup/patchset/meta`:

```
# cd <package_name>
# chmod +x setup.sh setup/combo-layer # ./setup.sh
```

7. Verify that the following directory structure is present, at a minimum:

```
pwd: ~/<package_name>
|-- bitbake
|
|-- meta-intel
|    |
|    |--meta-isg
|--meta-openembedded
|--meta-virtualization
|--meta-yocto
```

8. Run the script to build the environment essentials:

```
# cd ~/<package_name>
# source oe-init-build-env build
```

This also changes the current working directory to /<package_name>/build.

Note: Be sure to source the file while in /poky since the build directory is created based on the current working directory.

9. By default, the 64-bit version of the OS is built. If the 32-bit OS is required, edit the file `~/<package_name>/build/conf/local.conf` and update the MACHINE line as follows: `MACHINE ?= "intel-core2-32"`.
10. Build the SDK image by running the commands:

```
# cd ~/<package_name>/build
# bitbake core-image-sato-sdk
```

Note: The `bitbake` cannot run as a root user. Note the following:

- If build errors appear to be script errors such as the one shown in the following example, verify that /bin/sh is linked to /bin/bash as described in Step 7.

  ```
  [: 128: cmdline: unexpected operator |
  ```

- If the build fails and `bitbake` command needs to be executed again, repeat Step 7 to source `oe-init-build-env`.

  Log files (including errors and warnings) for the build are included in the `~/<package_name>/build/tmp/log directory`.

- Warning messages may be observed during the build process. These can be safely ignored.
- If an error is returned stating `bitbake` is not installed, verify that you sourced the `oe-init-build-env` file in `~/<package_name>` as described in Step 7.
- The command may take several hours to complete, depending on the particular software development machine and network speed.

11. Verify that the `hddimg` is created in `~/<package_name>/build/tmp/deploy/images/<intel-corei7-64 | intel-core2-32>`. The 64-bit version is titled `core-image-sato-sdk-intel-corei7-64.hddimg`.

5.2 Creating the Linux* Boot Disk

Note: If you are upgrading the acceleration drivers, proceed to Section 5.4, "Upgrading Acceleration Software".

5.2.1 Locating the hddimg

If creating your own `hddimg` via the process in Section 5.1, "Building the Yocto* SDK Image", your `hddimg` is located in `~/<package_name>/build/tmp/ deploy/images/<intel-corei7-64 | intel-core2-32>`.

5.2.2 Creating the Boot Disk

Note: Special care must be taken when creating the boot disk, since any misidentification of the target disk can overwrite critical data. Back up your data if there is any doubt about which disk you will be writing to in the following steps.

A script file is included in the Yocto* BSP for creating the disk image. The script is called `mkefidisk.sh` and is located in the following directory:
`~/<package_name>/scripts/contrib`

Usage is:
`mkefidisk.sh HOST_DEVICE image.hddimg TARGET_DEVICE`

where:

`HOST_DEVICE =>` Device to install image to.

`TARGET_DEVICE =>` Name of the device as target device sees it. This would likely be `/dev/sds`.

1. Identify the device name for your HOST_DEVICE. This is the drive the Yocto* image is being installed to. Study the output of one or more of the following commands to give confidence as to which disk is which:
   ```
   # sudo parted -l # df -h
   # cat /proc/partitions
   ```
2. Launch the `mkefidisk.sh` script using the HOST_DEVICE identified in Step 1, the hard drive image created in Section 5.1 or pre-built image, and the TARGET_DEVICE (likely `/dev/sda`). The following commands show the command for building the target image to the `/dev/sdb` drive with the 64-bit image created in Section 5.1.
   ```
   # cd ~<package_name>
   # sudo ./scripts/contrib/mkefidisk.sh /dev/sdb
   ./build/tmp/deploy/images/ intel-corei7-64/core-image-sato-sdk-intel-corei7-64.hddimg /dev/sda
   ```
   Answer "**y**" when prompted to prepare the EFI image on the `HOST_DEVICE (/dev/ sdb` in this example).

   During the process, an error dialog may appear that states Unable to open a folder for 3.3 GB Volume or Unable to open a folder for ROOT. These errors can be safely ignored. Click **OK** to close the error dialog.

   When the image preparation is complete, the following message appears:
   `Installation completed successfully`
3. Power down the build system when the image preparation is complete. Insert the newly created SATA drive or USB stick into the target system and boot to the Yocto* OS.

5.3 Additional Information on Software

Note: If you are upgrading the acceleration drivers, proceed to Section 5.4, "Upgrading Acceleration Software".

The software package contains a set of sample tests that exercises acceleration functionality. This section describes the steps required to build and execute the sample tests.

The sample application is provided for the user space.

5.3.1 Loading the Sample Code

The acceleration kernel module must be installed, and the software must be started before attempting to execute the sample code. This can be verified by running the following commands:

```
# lsmod | grep "qa"
# /etc/init.d/qat_service status
```

Typical output is similar to the following:

```
~# /etc/init.d/qat_service status Checking status of all devices. There
is 1 QAT acceleration device(s) in the system:
qat_dev0 - type: c3xxx, inst_id: 0, bsf: 01:00:0, #accel: 2 #engines: 6
state: up
```

Install the memory driver:

```
# insmod /lib/modules/4.1.8-yocto-standard/updates/drivers/crypto/qat/
usdm_drv.ko
```

The sample application is executed by launching the application for user space. The application is located in the /usr/bin directory. Section 4.1.2, "Loading the Sample Code" includes a list of run-time parameters for the application. To execute the sign of life test, use the following command:

```
# ./cpa_sample_code signOfLife=1
```

5.4 Upgrading Acceleration Software

This section describes the steps required to update the existing Yocto* image with the updated acceleration software. It enables usage of the newer acceleration software package without rebuilding the target image.

1. Perform the following commands to install kernel header files:

   ```
   # cd /usr/src/kernel/
   # make oldconfig && make modules_prepare && make scripts
   # ln -s /usr/src/kernel /lib/modules/4.4.13-yocto-standard/build
   ```

2. Create a working directory for the software. This directory can be user defined, but for the purposes of this document a recommendation is provided:

   ```
   # mkdir /QAT && cd /QAT
   ```

Note: In this document, the working directory is assumed to be `/QAT`. This directory is the `ICP_ROOT`.

3. Transfer the tarball to the development board using any preferred method, for example, the USB memory stick, CDROM, or network transfer in the /QAT directory.

Unpack the tarball using the following command:

```
# tar -zxof <QAT tarball name>
```

4. Restricting access to the files is recommended:

```
# chmod -R o-rwx *
```

5. Uninstall the existing acceleration software using the command:

```
# make uninstall && make clean && make distclean
```

6. Install the acceleration software and acceleration sample code using the following commands:

```
# ./configure
# make -j install
# make -j samples-install
```

Note: Using the "-j" simultaneous job option may cause compile issues on some distro's e.g. Fedora 40.

7. Execute sample code:

```
# cd build
# export LD_LIBRARY_PATH=/usr/lib64
# ./cpa_sample_code
```

Refer to Section 4.1.2, "Loading the Sample Code" for list of run-time parameters for the application.

§

6 Physical Function to Virtual Function Communications

In a virtualized environment, a Physical Function (PF) device driver runs on the host and Virtual Function (VF) drivers on the guests. The PF driver initializes the device andloads firmware.

The PF driver communicates with the VF drivers to exchange information and state. Information communicated to the VF driver includes:

- Device capabilities. Capabilities may be filtered by fuses, soft straps, or firmware.
- Driver compatibility. Different driver versions may be running on host and guest. Driver versions may not always be compatible. New features may be developed which, when enabled on the PF, introduce an incompatibility with older VF drivers. If the drivers are incompatible, the VF driver does not complete initialization.
- Event notification. Events, such as errors, which are detected on the PF may need to send notifications to the VFs.

§

Appendix A Avoiding Kernel Crashes with PCH Intel® SKUs

Note: Some Linux* versions, including RHEL/ CentOS* 7.3, will not boot/install with PCH Intel® QAT SKUs (E/M/T/L). This is due to a software driver bug in the in-kernel drivers for Intel® QAT. Pre-QS SKUs and 1G/2/4 SKUs are not affected.

Note: Intel® is committed to moving away from non-inclusive terms such as 'blacklist' and will do so at the soonest possible opportunity, governed by dependencies on other software, such as `modprobe`.

To avoid the problem, change the grub boot options to block the loading of the Intel® in-kernel driver for PCH (qat_c62x) at installation time and for future boots until the driver is updated and/or the in-kernel qat_c62x.ko file is deleted.

Note: These instructions may not cover the case in which the kernel source is updated. Beforeupdating the kernel, be sure to understand if kernel crashes may result due to this in-kernel Intel® QAT issue.

Note: Before uninstalling an Intel® QAT package, ensure that the package will not restore an in-kernel `qat_c62x.ko` file that has the issue.

Note: This issue is present approximately from the Linux* kernel 4.5 to kernel 4.9 and derivations thereof. The fixes are documented here:

- https://patchwork.kernel.org/patch/9485107/
- https://patchwork.kernel.org/patch/9485109/

Note: Some Linux* kernels after 4.9 may also experience kernel crashes with `qat_c62x.ko` or other `qat` modules. Follow the same recommended procedures described below.

A.1 Recommended Procedures

A.1.1 Installing the Operating System

Follow these instructions to safely install the operating system:

1. Boot platform from installation source (DVD, CD, USB)
2. Select “Install CentOS* Linux* v7.3” (or equivalent) menu from the GRUB list (but don’t press Enter).
3. Enter the grub options edit mode. This may be done by pressing Tab or e.
4. Add `modprobe.blacklist=qat_c62x` to the boot options. If you are not sure where exactly to make the edit, you can just find the word `quiet` and change it to `modprobe.blacklist=qat_c62x.`
5. Press Enter to boot.
6. Continue with installation process. The platform reboots when installation is complete.

A.1.2 Operating System First Boot

At the next reboot select the kernel you want to boot from the GRUB list (but don't press Enter).

1. Press e.
2. Append to the kernel command line `modprobe.blacklist=qat_c62x` (line that starts with `linuxefi` or perhaps `linux16`). If you are not sure where exactly to make the edit, you can just find the word `quiet` and change it to `modprobe.blacklist=qat_c62x`.
3. Press Ctrl+x to boot.

Note: If you still see a kernel crash including the keywords `qat` or `adf`, refer to Section A.2, "Alternative Module Load Blocking Procedures" for additional ideas to reach a command prompt.

Upon reaching the command prompt, remove the in-kernel `qat_c62x.ko` file that can lead to a kernel crash when inserted into the kernel:

```
# rm /usr/lib/modules/`uname -
r`/kernel/drivers/crypto/qat/qat_c62x/qat_c62x.ko
```

This prevents the offending module from being reloaded in all cases except the case in which a new kernel is loaded.

Reboot and verify that no kernel crash is observed.

```
# shutdown -r now
```

A.2 Alternative Module Load Blocking Procedures

Depending on your operating system and environment, some alternative methods of module load blocking can be attempted or utilized in order to get to a command prompt or for avoiding the kernel crash in the long term.

Note: Some methods may have unintended consequences (e.g. out-of-kernel QuickAssist packages may not install).

A.2.1 Grub Options

Try these options one at a time:

```
qat_c62x.blacklist=yes

rdblacklist=qat_c62x

module_blacklist=qat_c62x
```

A.2.2 Changes to Configuration File

Append the following to /lib/modprobe.d/dist-blacklist.conf after booting into rescue mode:

```
blacklist intel_qat

blacklist qat_c62x

blacklist qat_dh895xcc
```

§