Getting Started Guide

Intel® QuickAssist Technology

Hardware Version 2.0

February 2023

Document Number: 632506

Performance varies by use, configuration and other factors. Learn more on the 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

1.1 About This Document

This getting started guide documents the instructions to obtain, build, install, and exercise the Intel® QuickAssist Technology (Intel® QAT) software for the Hardware Version 2.0 package.

In this document, for convenience:

- *Software package* is used as a generic term for the Intel® QAT Software Package for Hardware Version 2.0.
- *Acceleration driver* is used as a generic term for the software that allows the Intel® QAT Software Library APIs to access the Intel® QAT Endpoint(s).

Note: Refer to the Release Notes for a list of supported platforms.

1.2 Conventions and Terminology

The following conventions are used in this manual:

- `Code text` - code examples, command line entries, Application Porgramming Interface (API) names, parameters, filenames, directory paths, and executables.
- **Bold text** - graphical user interface entries, buttons, and actions in instructions.
- *Italic text* - key terms and publication titles.

The following terms and acronyms are used in this manual.

Table 1: Terminology

Term	Description
API	Application Programming Interface
asym	Asymmetric Cryptography
BDF	Bus Device Function

continues on next page

Table 1 – continued from previous page

Term	Description
BOM	Bill of Materials
CBC	Cipher Block Chaining
cy	Cryptography
dc	Data Compression
GRUB	Grand Unified Bootloader
OS	Operating System
PCI	Peripheral Component Interconnect
PF	PCIe Physical Function
Intel® QAT	Intel® QuickAssist Technology
SKU	Stock Keeping Unit
sIOV	Scalable IOV
SR-IOV	Single Root-I/O Virtualization
VF	Virtual Function

1.3 Features Implemented

Implemented features are listed in the Release Notes.

1.4 List of Files in Release

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

1.5 Package Release Structure

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

Table 2: Package Release Structure

Files/Directory	Comments
`IntelQAT20<version>.tar.gz`	Top-level Intel® QAT package
`./filelist`	List of files in this package
`./config_guess`	Build and installer files
`./config.h.in`	
`./config.sh`	
`./config.sub`	
`./configure`	
`./install-sh`	
`./Makefile.in`	

continues on next page

Table 2 – continued from previous page

Files/Directory	Comments
./missing	
./LICENSE.GPL	License file
./versionfile	Version file
./quickassist	Top-Level acceleration softare directory
./README	README file with instructions on how to compile the driver

2 System Configuration

This section describes the process of configuring the system prior to the Intel® QuickAssist Technology (Intel® QAT) driver installation.

2.1 Configuring BIOS

Note: If installing the Intel QAT 2.0 driver for use in a virtual environment, refer to the Virtualization Deployment Guide for additional details.

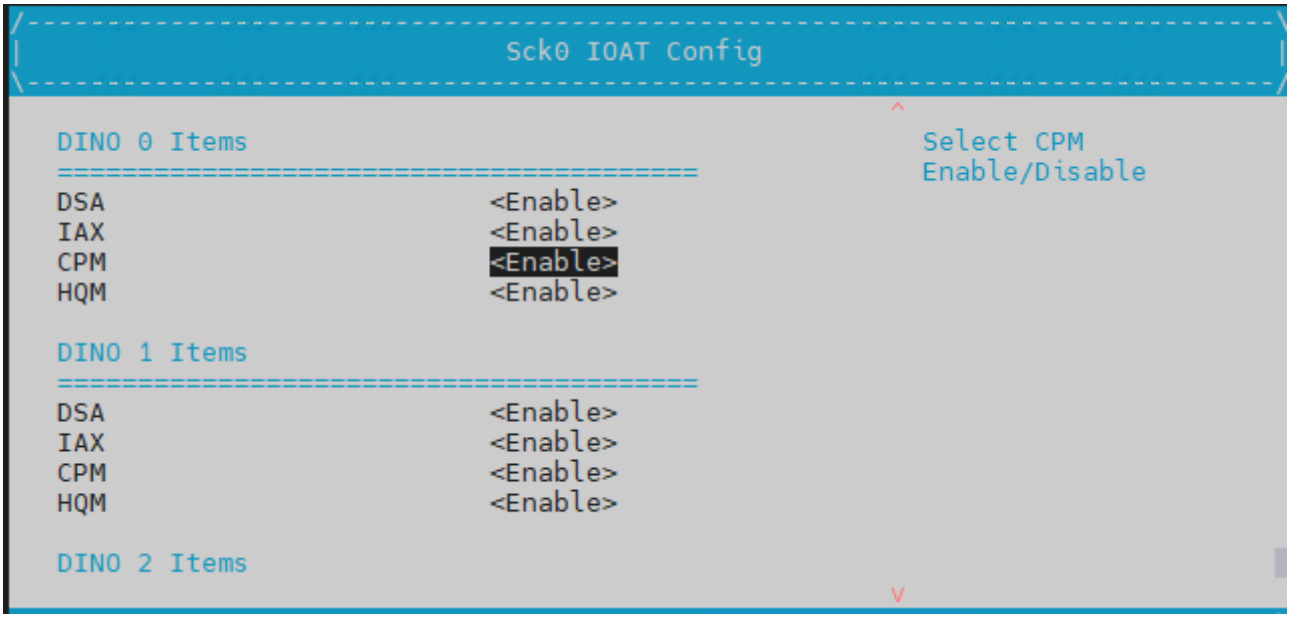
If BIOS updates are required, the following command can be used to reboot the system and enter the BIOS setup:

```
systemctl reboot --firmware-setup
```

2.2 Disabling QAT Endpoints

Depending on the hardware SKU, there can be up to 4 QAT endpoints per socket. It is possible to disable individual QAT endpoints by following the instructions below:

1. **Enter** BIOS setup.
2. **Navigate** to the following path where `<n>` corresponds to the socket containing the QAT endpoint(s) to be disabled: `EDKII Menu > Socket Configuration > IIO Configuration > IOAT Configuration > Sck<n> > IOAT Configuration`
3. **Update** the *CPM* value to *Disable* for each QAT endpoint to be disabled for each socket.
4. **Save** changes.
5. **Reboot** the system.



2.3 Configuring Operating System

There are a few configuration items that may need to be completed, such as updating `dnf` or `apt` configuration files as well as the system security configuration. This section describes these items.

2.3.1 Updating dnf Configuration Files

Important: This section is optional for RPM-based Linux distributions such as RHEL*, CentOS*, and Fedora*.

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

1. **Add** a line similar to the following in the `/etc/dnf/dnf.conf` file. The line can be added to the end of the file.

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

> **Note:** `<proxy_server:portnum>` is replaced with your server information. Contact your network administrator for details on the proxy server.

2. If your corporate proxy server requires a username and password, **specify** these by adding the following two settings in the `dnf.conf` file.

```
proxy_username=YOUR-PROXY-USERNAME-HERE
proxy_password=YOUR-SUPER-SECRET-PASSWORD-HERE
```

2.3.2 Updating apt Configuration Files

Important: This section is optional for DEB-based Linux distributions such as Ubuntu*.

`apt` is the default package manager for Debian* based distributions such as Ubuntu*. To use `apt` in a corporate network, the following updates may be required:

1. **Create** (or **edit** if the file already exists) a file named as `apt.conf` in the `/etc/apt` directory.

```
sudo nano /etc/apt/apt.conf
```

2. **Add** the following lines to the `apt.conf` file:

```
Acquire::http::Proxy "http://[YOUR-PROXY-USERNAME-HERE]:[YOUR-SUPER-SECRET-
↪PASSWORD-HERE]@ [proxy-web-or-IP-address]:[port-number]";
Acquire::https::Proxy "http://[YOUR-PROXY-USERNAME-HERE]:[YOUR-SUPER-SECRET-
↪PASSWORD-HERE]@ [proxy-web-or-IP-address]:[port-number]";
```

Note: `YOUR-PROXY-USERNAME-HERE` and `YOUR-SUPER-SECRET-PASSWORD-HERE` are optional parameters.

3. **Save** the file and **exit**.
4. **Reboot** the system. The configuration wil be applied after a reboot.

2.3.3 Installing Package Dependencies

The Intel QAT package depends on a number of libraries that must be installed first on the system.

2.3.3.1 RPM-based package dependencies

Important: This section is required for RPM-based Linux distributions such as RHEL*, CentOS*, and Fedora*.

1. **Enable** PowerTools Repository (may be required for yasm package).

```
sudo dnf -y install dnf-plugins-core
sudo dnf upgrade
sudo dnf config-manager --set-enabled powertools
```

2. **Install** the RPM-based package dependencies:

```
sudo dnf groupinstall "Development Tools"
sudo dnf install -y systemd-devel
sudo dnf install -y pciutils
sudo dnf install -y libudev-devel
sudo dnf install -y readline-devel
sudo dnf install -y libxml2-devel
sudo dnf install -y boost-devel
sudo dnf install -y elfutils-libelf-devel
sudo dnf install -y python3
sudo dnf install -y libnl3-devel
sudo dnf install -y kernel-devel-$(uname -r)
sudo dnf install -y gcc
sudo dnf install -y gcc-c++
sudo dnf install -y yasm
sudo dnf install -y zlib
sudo dnf install -y openssl-devel
sudo dnf install -y zlib-devel
sudo dnf install -y make
```

2.3.3.2 DEB-based package dependencies

Important: This section is required for DEB-based Linux distributions such as Ubuntu*.

1. **Install** the DEB-based package dependencies:

```
sudo apt-get update
sudo apt-get install -y libsystemd-dev
sudo apt-get install -y pciutils-dev
sudo apt-get install -y libudev-dev
sudo apt-get install -y libreadline6-dev
sudo apt-get install -y pkg-config
sudo apt-get install -y libxml2-dev
sudo apt-get install -y pciutils-dev
sudo apt-get install -y libboost-all-dev
sudo apt-get install -y libelf-dev
sudo apt-get install -y libnl-3-dev
sudo apt-get install -y kernel-devel-$(uname -r)
sudo apt-get install -y build-essential
sudo apt-get install -y yasm
sudo apt-get install -y zlib1g-dev
sudo apt-get install -y libssl-dev
```

2.3.4 System Security Considerations

Note:

- Specific OS/filesystem topics are outside of the scope of this document. For more information, refer to the Programmer's Guide.
- This section contains a high-level list of system security topics. This is not an exhaustive list.

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

- Employ effective security policies and tools; for instance, SELinux* is configured correctly and is active.
- Run and configure the firewall(s).
- Prevent privilege escalation at boot (including recovery mode); for instance, set a grub password. Additional details are described below.
- Remove unnecessary software packages.
- Patch software in a timely manner.
- Monitor the system and the network.
- Configure and disable remote access, as appropriate.
- Disable network boot.
- Require secure passwords.
- Encrypt files, up to full-disk encryption.
- Ensure physical security of the system and the network.
- Use mlock to prevent swapping sensitive variables from RAM to disk.
- Zero out sensitive variables in RAM.

3 Software Installation

This section provides details on building and installing the software.

Note: This document describes the steps required to install the out-of-tree acceleration software package. For details on installing the upstreamed acceleration software, refer to the installation and readme instructions at the Intel QuickAssist Technology Library (QATlib) repository.

3.1 Installation Overview

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

- Create the kernel module files and copy them the appropriate directory (e.g. `/usr/lib/modules/KERN_VERSION/kernel/drivers/crypto`)
- 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).

3.2 Unpacking the Software

The software package comes in the form of a tarball.

Note:

- The instructions in this document assume that you have super user privileges.
- In this document, the working directory is assumed to be `/QAT`. This directory is the `ICP_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.

```
export ICP_ROOT=/QAT
mkdir -p $ICP_ROOT
cd $ICP_ROOT
```

2. **Transfer** the tarball to the system in the `$ICP_ROOT` directory. **Unpack** the tarball using the following command:

```
tar -zxof QAT20.L.*.tar.gz
```

3. Restricting access to the files is recommended.

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

3.3 Configure Acceleration Software

Note:

- If installing the acceleration software for use in a virtual environment refer to the Virtualization Deployment Guide for additional details.
- The `./configure` script handles many options that may be of interest. For instance, there is a wide range of possible configurations, including build or install virtualization support (host or guest) or no virtualization support. Some build options may need to be passed as a parameter to the `./configure` script before proceeding with the installation. A complete list of configuration options is available in *Configuration Options*.

1. **Prepare** the package installation by checking the prerequisites and configuring the build options by running a script using the following command:

```
./configure
```

A welcome message is displayed, followed by the configured build options. Successfull configuration will look similar to the following:

```
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
...
checking for kernel sources... yes
checking that generated files are newer than configure... done
configure: creating ./config.status
config.status: creating Makefile
config.status: creating config.h
config.status: executing depfiles commands

================================================================================
Type make followed by make install to build QAT with the following options
================================================================================
    USE_HARD_CODED_PRIMES [0]
    ICP_ARCH_USER [x86_64]
    ICP_BUILDSYSTEM_PATH [/root/QAT/quickassist/build_system]
    ICP_BUILD_OUTPUT [/root/QAT/build]
    ICP_ENV_DIR [/root/QAT/quickassist/build_system/build_files/env_files]
    ICP_TOOLS_TARGET [accelcomp]
    MAX_MR [50]
    KERNEL_SOURCE_ROOT [/lib/modules/4.18.0-193.el8.x86_64/build]
    ICP_DEBUG [false]
    QAT_UIO [false]
    ICP_PARAM_CHECK [false]
    ICP_DC_DYN_NOT_SUPPORTED [false]
    DISABLE_STATS [false]
    DRBG_POLL_AND_WAIT [false]
    ICP_LOG_SYSLOG [false]
    ICP_NONBLOCKING_PARTIALS_PERFORM [false]
    ICP_TRACE [false]
    ICP_DC_ONLY [false]
    ICP_DC_RETURN_COUNTERS_ON_ERROR [false]
    ICP_DISABLE_INLINE [false]
    INLINE [false]
================================================================================
```

3.3.1 Dependencies

Dependencies are required, see configure output. `yasm`, for example, may need to be manually downloaded and installed via https://yasm.tortall.net/Download.html . (`wget` latest URL or http://www.tortall.net/projects/yasm/releases/yasm-1.3.0.tar.gz ; `tar -zxvf yasm-1.3.0.tar.gz` , `cd yasm-1.3.0/` , `./configure`, `make && make install` . Then return to QAT configure/install commands.

3.3.2 Configuration Options

A complete list of compile flags and build parameters can also be obtained by executing the following command in the shell:

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

Note: Compiler flags to produce safer binaries are enabled by default.

Table 3: Compile Flag Options

Compile Flag	Description
`--disable-option-checking`	Ignore unrecognized –enable/–with option.
`--disble-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`).
`--enable-maintainer-mode`	Enable make rules and dependencies not useful (and sometimes confusing) to the casual installer.
`--enable-dependency-tracking`	Do not reject slow dependency extractors.
`--disable-dependency-tracking`	Speeds up one-time build.
`--enable-icp-debug`	Enables debugging.
`--enable-qat-uio`	Enables Userspace I/O.
`--disable-param-check`	Disables parameters checking in the top-level APIs (use for performance optimization).
`--disable-dc-dyn`	Disables dynamic compression support.
`--disable-stats`	Disables statistic collection (use for performance optimization).
`--enable-drbg-poll-and-wait`	Modifies the behavior of DRBG HT functions to use single threaded operation.
`--enable-icp-log-syslog`	Enables debugging messages to be outputted to the system log instead of standard output.
`--enable-icp-nonblocking-partials-perform`	Partial operations results are not being blocked.
`--enable-icp-sriov`	Enables Single-root I/O Virtualization in the QAT driver (available options: `host`, `guest`).

continues on next page

Table 3 – continued from previous page

Compile Flag	Description
`--enable-icp-trace`	Enables tracing for the Cryptography API.
`--enable-icp-asym-only`	Enables driver to support Asymmetric Crypto services only.
`--enable-icp-sym-only`	Enables driver to support Symmetric Crypto services only.
`--enable-icp-dc-only`	Enables driver supports only compression service (can optimize size of build objects).
`--enable-icp-dc-sym-only`	Enables driver 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.
`--disable-icp-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`.
`--enable-inline`	Enables `INLINE` feature.
`--enable-icp-hb-fail-sim`	Enable Heartbeat Failure Simulation.
`--enable-qat-coexistence`	Enables legacy and upstream driver coexistence.
`--enable-qat-lkcf`	Enables QAT registration with Linux Kernel Crypto Framework.
`--enable-qat-kpt-debug-key`	Enable QAT debug issue certificate.
`--disable-dc-strict-mode`	Disables Compress and Verify (CnV) functionality. See *below* for details.
`--enable-dc-error-simulation`	Enables Data Compression Error Simulation.

Important: The Compress and Verify feature checks and ensures data integrity in the compression operation of the Intel® QAT Data Compression API. This feature introduces an independent capability to verify the compression transformation.

Intel recommends that customers use the Compress and Verify capabilities for Intel® QAT compression operations.

As Compress and Verify provides an integrity check of the data, Intel cannot guarantee integrity of data that bypasses the Compress and Verify capability.

Intel does **not** support disabling Compress and Verify.

3.4 Install Acceleration Software

Note: It is recommended to *uninstall* previous installations of the acceleration software (if previously installed).

1. **Open** a terminal window and **switch** to superuser. Provide root password when prompted.

```
su
cd $ICP_ROOT
```

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

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

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

3. **Verify** the acceleration software kernel objects are loaded and ready to use with this command:

```
lsmod | grep qat
```

Depending on the specific hardware present, this command will return an output similar to the following:

```
qat_4xxx               45056  0
intel_qat             331776  2 qat_4xxx,usdm_drv
uio                    20480  1 intel_qat
mdev                   20480  2 intel_qat,vfio_mdev
vfio                   36864  3 intel_qat,vfio_mdev,vfio_iommu_type1
irqbypass              16384  2 intel_qat,kvm
```

Note: Not all modules will be required depending on the specific hardware present.

3.5 Uninstall Acceleration Software

1. **Open** a terminal windows and **switch** to superuser. Provide root password when prompted.

```
su
cd $ICP_ROOT
```

2. **Enter** the following commands to uninstall the acceleration software:

```
make uninstall
make clean
```

3.6 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
```

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
```

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

3.7 Configuration Files

When the acceleration software loads, it is configured based on the settings in the platform-specific configuration files.

The configuration files are in the `/etc` directory. Specifically:

- The name for the first configuration file for Intel® QuickAssist Technology Hardware Version 2.0 devices is `4xxx_dev0.conf`
- The name of the first configuration file for Intel® Communications Chipset 8925 to 8955 Series devices is `dh895xcc_dev0.conf`.

- The first configuration file for the Intel® C62x Chipset or Intel® Xeon® Processor D Family SoC is `c6xx_dev0.conf`.
- The first configuration file for Intel® Atom® C3000 Processor SoC is `c3xxx_dev0.conf`.
- The first configuration file for other Intel® Xeon® Processor D SoC platforms is `d15xx_dev0.conf`.

Note: If more than one device of a given type is present, its name includes `dev1`, `dev2`, etc.

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.

```
service qat_service restart
```

The software package includes multiple types of platform-specific 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.

Important: The software package has been validated with the default configuration files. Changes to the configuration files could have adverse effects.

Refer to the Programmer's Guide for additional information on the configuration files.

3.8 Running Applications as Non-Root User

The installation of Intel® QAT software package configures the driver to allow applications to run as non-root user. The users must be added to the `qat` group.

When the `make install` command is performed at the directory where the Intel® QAT package is installed, the following *udev* file is created which is responsible for setting up non-root access.

```
KERNEL=="qat_adf_ctl" MODE="0660" GROUP="qat" RUN+="/bin/chgrp qat/usr/local/bin/adf_ctl"
KERNEL=="qat_dev_processes" MODE="0660" GROUP="qat"

KERNEL=="usdm_drv" MODE="0660" GROUP="qat"

ACTION=="add", DEVPATH=="/module/usdm_drv" SUBSYSTEM=="module"
RUN+="/bin/mkdir / dev/hugepages/qat"

ACTION=="add", DEVPATH=="/module/usdm_drv" SUBSYSTEM=="module"
RUN+="/bin/chgrp qat /dev/hugepages/qat"

ACTION=="add", DEVPATH=="/module/usdm_drv" SUBSYSTEM=="module"
RUN+="/bin/chmod 0770 /dev/hugepages/qat"
```

(continues on next page)

(continued from previous page)

```
ACTION=="remove", DEVPATH=="/module/usdm_drv" SUBSYSTEM=="module"
RUN+="/bin/rmdir

/dev/hugepages/qat"

KERNEL=="uio*", ATTRS{vendor}=="0x8086", ATTRS{device}=="0x0435"
MODE="0660" GROUP="qat"

KERNEL=="uio*", ATTRS{vendor}=="0x8086", ATTRS{device}=="0x0443"
MODE="0660" GROUP="qat"

KERNEL=="uio*", ATTRS{vendor}=="0x8086", ATTRS{device}=="0x37c8"
MODE="0660" GROUP="qat"

KERNEL=="uio*", ATTRS{vendor}=="0x8086", ATTRS{device}=="0x37c9"
MODE="0660" GROUP="qat"

KERNEL=="uio*", ATTRS{vendor}=="0x8086", ATTRS{device}=="0x6f54"
MODE="0660" GROUP="qat"

KERNEL=="uio*", ATTRS{vendor}=="0x8086", ATTRS{device}=="0x6f55"
MODE="0660" GROUP="qat"

KERNEL=="uio*", ATTRS{vendor}=="0x8086", ATTRS{device}=="0x19e2"
MODE="0660" GROUP="qat"

KERNEL=="uio*", ATTRS{vendor}=="0x8086", ATTRS{device}=="0x19e3"
MODE="0660" GROUP="qat"
```

The updates to the udev rules are performed during the installation of the Intel® QAT driver.

The following steps need to be manually applied:

1. **Change** the amount of max locked memory for the username included in the `qat` group (the default memory limit is 64). This can be done by specifying the limit in: `/etc/security/limits.conf`.

   ```
   @qat - memlock 4096
   ```

4 Sample Applications

The software package contains a performance sample as well as functional sample applications. This section describes the steps required to build and execute these applications.

4.1 Performance Sample Code

The sample application is provided for the user space.

4.1.1 Compiling the Performance Sample Code

Note:

- These instructions assume the software package was untarred in the `$ICP_ROOT` directory.
- For details on running user space applications as non-root user refer to the section *Running Applications as Non-Root User*.

1. **Open** a terminal window and **switch** to superuser. Provide root password when prompted.

```
su
```

2. **Switch** to the `$ICP_ROOT` directory and **compile** the installation samples.

```
cd $ICP_ROOT
make samples-install
```

This compiles the acceleration sample code for *user space*. It also compiles the memory mapping driver used with the user space application.

4.1.2 Default Configuration Files

By default, the QAT configuration files enable asymmetric crypto and data compression services. If symmetric crypto is desired, the service must be enabled in the QAT configuration file. The QAT configuration files are included in `/etc` folder and are named `4xxx_dev<x>.conf` where `x` is the device number.

In this file, **replace** the line:

```
ServicesEnabled = asym;dc
```

With:

```
ServicesEnabled = sym;dc
```

4.1.3 Loading the Sample Code Application

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 `$ICP_ROOT/build`.

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"
service qat_service status
```

Typical output is similar to the following:

```
$ lsmod | grep "qa"
qat_4xxx               61440  0
intel_qat             401408  2 qat_4xxx,usdm_drv
uio                    20480  1 intel_qat
irqbypass              16384  4 intel_qat,vfio_pci_core,idxd_mdev,kvm

$ service qat_service status
Checking status of all devices.
There is 8 QAT acceleration device(s) in the system:
qat_dev0 - type: 4xxx,  inst_id: 0,  node_id: 0,  bsf: 0000:6b:00.0,  #accel: 1 #engines: 9
↪state: up
qat_dev1 - type: 4xxx,  inst_id: 1,  node_id: 0,  bsf: 0000:70:00.0,  #accel: 1 #engines: 9
↪state: up
qat_dev2 - type: 4xxx,  inst_id: 2,  node_id: 0,  bsf: 0000:75:00.0,  #accel: 1 #engines: 9
↪state: up
qat_dev3 - type: 4xxx,  inst_id: 3,  node_id: 0,  bsf: 0000:7a:00.0,  #accel: 1 #engines: 9
↪state: up
qat_dev4 - type: 4xxx,  inst_id: 4,  node_id: 1,  bsf: 0000:e8:00.0,  #accel: 1 #engines: 9
↪state: up
```

(continues on next page)

(continued from previous page)

```
qat_dev5 - type: 4xxx,  inst_id: 5,  node_id: 1,  bsf: 0000:ed:00.0,  #accel: 1 #engines: 9
↪state: up
qat_dev6 - type: 4xxx,  inst_id: 6,  node_id: 1,  bsf: 0000:f2:00.0,  #accel: 1 #engines: 9
↪state: up
qat_dev7 - type: 4xxx,  inst_id: 7,  node_id: 1,  bsf: 0000:f7:00.0,  #accel: 1 #engines: 9
↪state: up
```

Note: If the modules are not returned from the first command, refer to the *installation instructions* for additional information on starting the acceleration software.

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

```
./build/cpa_sample_code
```

4.1.4 Sample Code Parameters

The application allows the run-time parameters listed below:

Table 4: Sample Code Parameters

Parameter	Description
`cyNumBuffers=w`	Number of buffers submitted for each iteration. (default=20)
`cySymLoops=x`	Number of iterations of all symmetric code tests. (default= 5000)
`cyAsymLoops=y`	Number of iterations of all asymmetric code tests. (default=5000)
`runTests=1`	Run symmetric code tests.
`runTests=2`	Run RSA test code.
`runTests=4`	Run DSA test code.
`runTests=8`	Run ECDSA test code.
`runTests=16`	Run Diffie-Hellman code tests.
`runTests=32`	Run compression code tests.
`runTests=63`	Run all tests except the chained hash and compression tests. (default)
`runTests=128`	Run chained hash and compression test code.
`runStateful=1`	Enable stateful compression tests. Applies when compression code tests are run.
`signOfLife=1`	Indicates shorter test run that verifies the acceleration software is working. This parameter executes a subset of sample tests. Details are included in *signOfLife Test Parameter*. (default=0)
`getLatency=1`	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. Performance sample code needs to be compiled with flag to enable latency measurements. `make samples-install LATENCY_CODE=1`

continues on next page

Table 4 – continued from previous page

Parameter	Description
`getOffload-Cost=1`	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.
`includeLZ4=1`	Include LZ4 compression tests. Applies when compression code tests are run.

4.1.4.1 signOfLife Test Parameter

The `signOfLife` parameter is used to specify that a subset of the sample tests are executed with smaller iteration counts. This provides a quick test to verify 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.

4.1.4.2 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
```

4.1.5 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 Functional Sample Applications

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

For more details, refer to the Intel® QuickAssist Technology API Programmer's Guide.

4.2.1 Compiling the Acceleration Functional Sample Code

Note: These instructions assume the software package has been untarred to the `$ICP_ROOT` directory.

The acceleration functional sample code can be compiled manually.

1. **Compile** for the user space using the following commands:

```
cd $ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/functional
make all
```

The generated sample applications are located at: `$ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/functional/build`

4.2.2 Executing the Acceleration Functional Sample Code in User Space

1. To execute the acceleration functional sample code in user space, **use** a command similar to the following:

```
cd $ICP_ROOT/quickassist/lookaside/access_layer/src/sample_code/functional/build
./hash_file_sample
```

Note: The `hash_file_sample` is one of the functional user space applications. You can launch the other user space applications in a similar fashion.