# Intel® Data Plane Development Kit - Intel® QuickAssist Technology Sample Application

## User Guide

*April 2012*

**Intel Confidential**

Reference Number: 497691-1.0

# Contents

**1.0 Introduction** .......... 4
**2.0 Overview** .......... 4
2.1 Setup .......... 5
**3.0 Building the Application** .......... 5
**4.0 Running the Application** .......... 6
4.1 Intel® QuickAssist Technology Configuration Files .......... 6
4.2 Traffic Generator Setup and Application Startup .......... 6

## List of Figures

1 Intel® QuickAssist Technology Application Block Diagram .......... 4

# Revision History

| Date | Revision | Description |
|---|---|---|
| April 2012 | 1.0 | Initial version of document |

# 1.0 Introduction

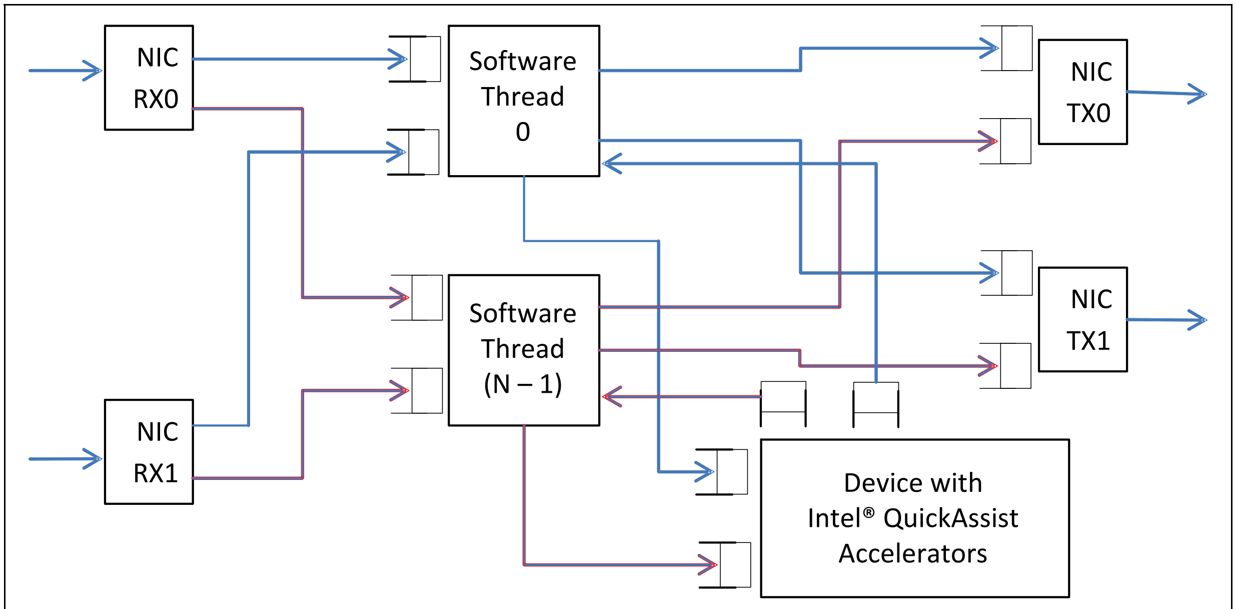
The current sample application demonstrates the use of the cryptographic operations provided by the Intel® QuickAssist Technology from within the Intel® Data Plane Development Kit (Intel® DPDK) environment. Therefore, building and running this application requires having both the Intel® DPDK and the QuickAssist Technology Software Library installed, as well as at least one Intel® QuickAssist Technology hardware device present in the system.

For this sample application, there is a dependency on the *Crystal Forest Software for Linux* with Cave Creek B0 Silicon* package, which can be downloaded from: http://downloadcenter.intel.com/Detail_Desc.aspx?agr=Y&DwnldID=20848

# 2.0 Overview

An overview of the application is provided in Figure 1. For simplicity, only two NIC ports and one Intel® QuickAssist Technology device are shown in this diagram, although the number of NIC ports and Intel® QuickAssist Technology devices can be different.

**Figure 1. Intel® QuickAssist Technology Application Block Diagram**



**Note**: Lines in blue show the packet flow for Software Thread 0, and lines in red show the packet flow for Software Thread (N – 1).

The application allows the configuration of the following items:

- Number of NIC ports
- Number of logical cores (lcores)
- Mapping of NIC RX queues to logical cores

Each lcore communicates with every cryptographic acceleration engine in the system through a pair of dedicated input – output queues. Each lcore has a dedicated NIC TX queue with every NIC port in the system. Therefore, each lcore reads packets from its NIC RX queues and cryptographic accelerator output queues and writes packets to its NIC TX queues and cryptographic accelerator input queues.

Each incoming packet that is read from a NIC RX queue is either directly forwarded to its destination NIC TX port (forwarding path) or first sent to one of the Intel® QuickAssist Technology devices for either encryption or decryption before being sent out on its destination NIC TX port (cryptographic path).

The application supports IPv4 input packets only. For each input packet, the decision between the forwarding path and the cryptographic path is taken at the classification stage based on the value of the IP source address field read from the input packet. Assuming that the IP source address is A.B.C.D, then if:

- D = 0: the forwarding path is selected (the packet is forwarded out directly)
- D = 1: the cryptographic path for encryption is selected (the packet is first encrypted and then forwarded out)
- D = 2: the cryptographic path for decryption is selected (the packet is first decrypted and then forwarded out)

For the cryptographic path cases (D = 1 or D = 2), byte C specifies the cipher algorithm and byte B the cryptographic hash algorithm to be used for the current packet. Byte A is not used and can be any value. The cipher and cryptographic hash algorithms supported by this application are listed in the `crypto.h` header file.

For each input packet, the destination NIC TX port is decided at the forwarding stage (executed after the cryptographic stage, if enabled for the packet) by simply selecting the adjacent port of the packet input port: out_port = in_port ^ 1 (`in_port` XOR `1`). For example, for input port 0, the output port is 1; for input port 1, the output port is 0, and so on.

For the cryptographic path, it is the payload of the IPv4 packet that is encrypted or decrypted.

## 2.1 Setup

Building and running this application requires having both the Intel® DPDK package and the QuickAssist Technology Software Library installed, as well as at least one Intel® QuickAssist Technology hardware device present in the system.

For more details on how to build and run Intel® DPDK and Intel® QuickAssist Technology applications, please refer to the following documents:

- *Intel® DPDK Getting Started Guide* (450248)
- *Crystal Forest Software for Linux* Getting Started Guide* (440005)

For more details on the actual platforms used to validate this application, as well as performance numbers, please refer to the following document:

- *Intel® DPDK Test Report* (450257)

# 3.0 Building the Application

Steps to build the application:

1. Set up the following environment variables:

```
export RTE_SDK=<Absolute path to the Intel DPDK installation folder>
export ICP_ROOT=<Absolute path to the Intel QAT installation folder>
```

2. Set the target (a default target is used if not specified). For example:

   ```
   export RTE_TARGET=x86_64-default-linuxapp-gcc
   ```

   Refer to the Intel® DPDK Getting Started Guide for possible RTE_TARGET values.

3. Build the application:

   ```
   cd ${RTE_SDK}/examples/dpdk_qat
   make
   ```

# 4.0 Running the Application

## 4.1 Intel® QuickAssist Technology Configuration Files

The Intel® QuickAssist Technology configuration files used by the application are located in the config_files folder in the application folder. There are two sets of configuration files available for different Intel Customer Reference Boards (CRBs):

- Stargo CRB (single CPU socket): located in the stargo_B0 folder
  - dh89xxcc_qa_dev0.conf
- Shumway CRB (dual CPU socket): located in shumway_B0 folder.
  - dh89xxcc_qa_dev0.conf
  - dh89xxcc_qa_dev1.conf

*Note:* The configuration files mentioned above are intended for CRBs with on-board accelerators only. For CRBs with plug-in cards (CPICs) that contain additional Intel® QuickAssist Technology accelerators, additional configuration files are required. See the *Cave Creek Plug-in Card (CPIC) User's Guide*.

The relevant configuration file(s) must be copied to the /etc/ directory.

Please note that any change to these configuration files requires restarting the Intel® QuickAssist Technology driver using the following commands:

```
/etc/init.d/adf_ctl down
/etc/init.d/adf_ctl up
```

Refer to the *Crystal Forest Software Programmer's Guide* and the *Crystal Forest Software for Linux* Getting Started Guide* for information on the Intel® QuickAssist Technology configuration files.

## 4.2 Traffic Generator Setup and Application Startup

The application has a number of command line options:

```
dpdk_qat [EAL options] -- -p PORTMASK [--no-promisc]
         [--config(port,queue,lcore)[,(port,queue,lcore]]
```

where,

- --p PORTMASK: Hexadecimal bitmask of ports to configure

- `--no-promisc`: Disables promiscuous mode for all ports, so that only packets with the Ethernet MAC destination address set to the Ethernet address of the port are accepted. By default promiscuous mode is enabled so that packets are accepted regardless of the packet's Ethernet MAC destination address.
- `--config(port,queue,lcore)[,(port,queue,lcore)]`: determines which queues from which ports are mapped to which cores.

Refer to the *L3 Forwarding Sample Application User Guide* (482251) for more detailed descriptions of the '--config' command line option.

As an example, to run the application with two ports and two cores, which are using different Intel® QuickAssist Technology execution engines, performing AES-CBC-128 encryption with AES-XCBC-MAC-96 hash, the following settings can be used:

- Traffic generator source IP address: 0.9.6.1
- Command line:

```
./build/dpdk_qat -c 0xff -n 2 -- -p 0x3 --config '(0,0,1),(1,0,2)'
```

Refer to the *Intel® DPDK Test Report* (450257) for more examples of traffic generator setup and the application startup command lines. If no errors are generated in response to the startup commands, the application is running correctly.

§ §