zlib* Sample Patch for Intel® QuickAssist Technology

Application Note

November 2014

Document Number: 330691-002

Contents

Tables

Revision History

Date	Revision	Description
November 2014	002	Updated Section 1.2.3, “Limitations” on page 7 Change bars indicate areas of change.
September 2014	1.0	Initial public release; was previously document number 477931

§ §

1.0 Introduction

1.1 About this Manual

This document discusses the following topics about the zlib* Sample Patch for Intel® QuickAssist Technology:

- features and limitations
- build and installation
- test application usage

In this document, for convenience:

- *Software package* is used as a generic term for the Intel® Communications Chipset 89xx Series Software package or the Intel® Communications Chipset 8925 to 8955 Series Software package.
- *Acceleration drivers* is used as a generic term for the software that allows the QuickAssist Software Library APIs to access the Intel® QuickAssist Accelerator(s) integrated in the Intel® Communications Chipset 8900 to 8920 Series or the Intel® Communications Chipset 8925 to 8955 Series PCH.

1.2 Software Overview

This section lists the features and limitations of the zlib* Sample Patch for Intel® QuickAssist Technology.

1.2.1 Features

This software enables access to the Intel® QuickAssist Technology implemented on Intel® Communications Chipset 8900 to 8920 Series based platforms and Intel® Communications Chipset 8925 to 8955 Series platforms using the zlib library APIs. The zlib software library and the Intel® QuickAssist Technology both implement the deflate algorithm as described in RFC 1951. Even though both implementations are compliant to RFC 1951, it is important to note that they are not equivalent in all cases.

The current version of the zlib Sample patch supports the co-existence of the zlib Sample patch and the libcrypto* (OpenSSL*) Sample Patch for Intel® QuickAssist Technology (see Table 4).

Table 1 lists zlib functions that are supported when hardware compression is enabled. Table 2 on page 6 lists zlib functions that are not supported when hardware compression/decompression is enabled.

Table 1. zlib Functions supported when compression is enabled

Function
`deflateInit`
`deflateInit2` • calling this function offloads deflate processing to the Intel® QuickAssist Technology's implementation • supports 8 K and 32 K window sizes; note, however, that the amount of history maintained between requests to the hardware depends on the compression level (for more details, see the appropriate programmer's guide shown in Table 4). • memLevel and strategy parameter values are not honored
deflate • supports `Z_NO_FLUSH`, `Z_FINISH`, and `Z_SYNC_FLUSH` flush flags • `Z_SYNC_FLUSH` immediately sends data to the acceleration driver rather than internal buffering within zlib library
deflateEnd
deflateReset
deflateBound
inflate • calling this function offloads inflate processing to the Intel® QuickAssist Technology's implementation • last request must be submitted with a `Z_FINISH` flush flag • supports only `Z_NO_FLUSH`, `Z_FINISH`, and `Z_SYNC_FLUSH` flush flags • `Z_SYNC_FLUSH` is used to disable buffering within the zlib library before sending to acceleration driver
`inflateEnd`
inflateInit
`inflateInit2` • with hardware decompression enabled there is no support for automatic gzip/zlib file detection • supports 8 K and 32 K window sizes
inflateReset
zlibVersion

Table 2. zlib Functions not supported when compression is enabled (Sheet 1 of 2)

Function	Behavior
deflateSetDictionary	`returns Z_OK`
deflateCopy	`returns Z_STREAM_ERROR`
deflateParams	`returns Z_OK`
deflateTune	`returns Z_OK`
deflatePending	`returns Z_OK`
deflatePrime	`returns Z_OK`
deflateSetHeader	`returns Z_OK`
inflateSetDictionary	`returns Z_STREAM_ERROR`
inflateSync	`returns Z_STREAM_ERROR`
inflateCopy	`returns Z_STREAM_ERROR`
inflateReset2	This function should not be called externally.
inflatePrime	`returns Z_STREAM_ERROR`
inflateMark	`returns -1L << 16`

Table 2. zlib Functions not supported when compression is enabled (Sheet 2 of 2)

Function	Behavior
inflateGetHeader	`returns Z_STREAM_ERROR`
inflateBackInit	undefined behavior - not currently stubbed
inflateBack	undefined behavior - not currently stubbed
inflateBackEnd	undefined behavior - not currently stubbed

Table 3 lists zlib functions that were added to the zlib interface to enable hardware compression/decompression. After applying the patch, the function definitions are defined in **`zlib.h`**.

Mux LM

Table 3. New zlib Functions to enable hardware compression/decompression

Function
zlibSetupEngine
zlibStartupEngine
zlibShutdownEngine

1.2.2 Mux Layer Support

The zlib*Sample Patch for Intel® QuickAssist Technology supports running with both Intel® Communications Chipset 8900 to 8920 Series Accelerators and Intel® Communications Chipset 8925 to 8955 Series Accelerators simultaneously via the Mux Layer.

As a prerequisite, the Intel® QuickAssist Technology Driver must have been built with Mux support enabled.

To build the patched zlib* to work with Mux you must export an environment variable with a command such as:

```
# export WITH_CPA_MUX=1
```

Note: Only define the environment variable if you need to use the Mux Layer.

1.2.3 Limitations

The zlib* Sample Patch for Intel® QuickAssist Technology has the following limitations:

- This software enables access to the Intel® QuickAssist Technology using the zlib library APIs. Even though the zlib software library and the Intel® QuickAssist Technology both implement a deflate-compliant (RFC 1951) compressor and decompressor, their implementations are not equivalent in all cases.
- This zlib library should not be installed as the main system zlib library via the `make install` command because when hardware compression/decompression is enabled, it does not support full system functionality.
- This software release uses static Huffman Trees.

Note: Review the README.txt file in the package for other known limitations.

1.3 Documentation

1.3.1 Where to Find Current Software and Documentation

Refer to Table 4 a list of all Intel® QuickAssist Technology software release and associated collateral.

All publicly released Intel® QuickAssist Technology software release associated collateral can be found by visiting the Intel® Open Source Technology Center at: https://01.org/packet-processing/intel%C2%AE-quickassist-technology-drivers-and-patches

1.3.2 Product Documentation

This release includes:

- zlib* Sample Patch for Intel® QuickAssist Technology Application Note (this document)

The documents listed in Table 4 may be accessed as described in Section 1.3.1.

Table 4. Related Documents

Document Name	Number
Intel® Communications Chipset 8900 to 8920 Series Software Programmer's Guide	330753[1]
Intel® QuickAssist Technology API Programmer's Guide	330684[1]
Intel® QuickAssist Technology Cryptographic API Reference Manual	330685[1]
Intel® QuickAssist Technology Data Compression API Reference Manual	330686[1]
Intel® QuickAssist Technology Performance Optimization Guide	330687[1]
libcrypto* (OpenSSL*) Sample Patch for Intel® QuickAssist Technology	477629[2]
libcrypto* (OpenSSL*) Sample Patch for Intel® QuickAssist Technology Application Note Note: This document is included in 477629.	[1]
Intel® Communications Chipset 89xx Series Software for Linux* - Release Notes (Version: QATmux.L.1.1.0-60 Release Notes)	330683[1]
Intel® Communications Chipset 89xx Series Software for Linux* Getting Started Guide	330750[1]
Intel® Communications Chipset 8925 to 8955 Series Software – Programmer's Guide	330683[1]
Intel® Communications Chipset 8900 to 8920 Series Performance Brief	330753[1]
Intel® Communications Chipset 8900 to 8920 Series Performance Brief	470252[2]
Intel® Communications Chipset 8925 to 8955 Series Performance Brief	527069[2]
zlib* Sample Patch for Intel® QuickAssist Technology (development branch) (zlib_quickassist_patch_L.0.4.7_001_devbranch.zip)	[1]
libcrypto* (OpenSSL*) Sample Patch for Intel® QuickAssist Technology (development branch) (libcrypto_quickassist_patch_l.0.4.6-008_devbranch.zip)	[1]

Notes:

1. All publicly released Intel® QuickAssist Technology software release associated collateral listed can be found by visiting the Intel® Open Source Technology Center at: https://01.org/packet-processing/intel%C2%AE-quickassist-technology-drivers-and-patches.
2. All Intel Confidential associated collateral listed can be found on the Intel Business Portal.

1.3.3 Documentation Conventions

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

1.4 Software Requirements

- Operating system: Fedora 16 64-bit version
- Kernel: GNU*/Linux* 3.1
- zlib 1.2.8
- Intel® Communications Chipset 89xx Series Software for Linux* version 1.5 or later, or Intel® Communications Chipset 8925 to 8955 Series Software for Linux* version 1.1 or later

2.0 Unpacking and Building the Software

This chapter provides instructions for unpacking and building the zlib* Sample Patch for Intel® QuickAssist Technology. The release consists of the patch file and the test application.

Note: Before installing the zlib* Sample Patch for Intel® QuickAssist Technology, you must install the acceleration software package for your platform. Refer to the Getting Started Guide for your platform for detailed instructions.

Note: System commands given in this chapter assume that the user is issuing commands from a bash shell. This is the default shell. Use the `echo $0` command to verify use of the bash shell or run `/bin/bash` to switch to the bash shell.

2.1 Installing zlib and Applying the Patch

Perform the following steps to install zlib and apply the zlib patch.

1. Open a terminal window and switch to superuser:
   ```
   # su
   # Password: <enter root password>
   ```
2. Set the `$ICP_ROOT` directory. By convention this will be:
   ```
   # export ICP_ROOT=/QAT/QAT1.6
   ```
   If the driver was compiled with support for the Intel® Communications Chipset 8900 to 8920 Series only, this variable may need to be set to one of the following:
   ```
   # export ICP_ROOT=/QAT/QAT1.5
   # export ICP_ROOT=/QAT
   ```
   If the following command does not return any files or directories, `$ICP_ROOT` was not set correctly:
   ```
   # ls $ICP_ROOT/quickassist
   ```
3. Create a folder named `$ICP_ROOT/quickassist/shims/zlib`. You may choose another directory name if desired. The recommended directory follows the convention of placing patches in a common location.
   ```
   # mkdir -p $ICP_ROOT/quickassist/shims/zlib
   ```
4. Change directories to the newly created directory:
   ```
   # cd $ICP_ROOT/quickassist/shims/zlib
   ```

5. Download the original zlib-1.2.8 software package.

```
# wget http://zlib.net/zlib-1.2.8.tar.gz
```

6. Extract the source files from the zlib software package to the `zlib` directory:

```
# tar xzof zlib-1.2.8.tar.gz
```

7. Extract the zlib* Sample Patch for Intel® QuickAssist Technology software package to the `zlib` directory:

```
# tar xzof <path_to>/zlib-1.2.8-qat.L.<version>.tar.gz
```

8. Apply the patch by performing the following command:

```
# cd zlib-1.2.8
# patch -p0 < ../zlib-1.2.8-qat.patch
```

9. Uninstall the Intel® QuickAssist Technology Memory Management kernel module (`qat_mem.ko`), if installed:

```
# rmmod qat_mem
```

10. Setup environmental variables:

```
# export ICP_BUILD_OUTPUT=$ICP_ROOT/build
# export ZLIB_ROOT=$ICP_ROOT/quickassist/shims/zlib/zlib-1.2.8
```

If and only if the Intel® QuickAssist Technology driver was built with mux layer support, enable this also in the zlib build:

```
# export WITH_CPA_MUX=1
```

11. Build the Intel® QuickAssist Technology Memory Management kernel module (`qat_mem.ko`):

```
# cd $ZLIB_ROOT/contrib/qat/qat_mem
# make
```

12. Install the memory kernel module:

```
# insmod ./qat_mem.ko
```

13. If compiling for use with Intel® Communications Chipset 8925 to 8955 Series hardware acceleration and not Intel® Communications Chipset 8900 to 8920 Series, set the ZLIB_DH895XCC variable:

```
# export ZLIB_DH895XCC=1
```

Note: If `WITH_CPA_MUX=1` is set, do not set `ZLIB_DH895XCC=1`.

14. Configure and build zlib by performing the following steps:

```
# cd $ZLIB_ROOT
# ./configure
# make
```

2.2 Test Applications

The zlib* Sample Patch for Intel® QuickAssist Technology package contains the following test applications:

- zpipe Application
- minigzip Application
- comptestapp Test Application

Note: As a reminder, setting the environment variable ZLIB_DH895xCC=1 or WITH_CPA_MUX=1 is required if using the Intel® Communications Chipset 8925 to 8955 Series. Otherwise, the applications will not use hardware acceleration.

2.2.1 Updating the Acceleration Configuration

Prior to running any of the following applications, updated configuration files must be in place and the acceleration service must be restarted. This only needs to be done on the initial run.

The configuration files included in the config directory listed below enable both Crypto and Compression services. If your application is only using the compression service, the crypto services should be disabled in the configuration file. This will increase overall compression performance because more memory will be available on the device for compression. This is done by updated the `ServicesEnabled` parameter in the `[GENERAL]` section.

For Intel® Communications Chipset 8900 to 8920 Series:

```
# cp $ICP_ROOT/quickassist/shims/zlib/zlib-1.2.8/contrib/qat/config/dh89xxcc/
multi_thread_optimized/dh89xxcc_qa_dev0.conf
/etc/dh89xxcc_qa_dev0.conf
# cp $ICP_ROOT/quickassist/shims/zlib/zlib-1.2.8/contrib/qat/config/dh89xxcc/
multi_thread_optimized/dh89xxcc_qa_dev0.conf
/etc/dh89xxcc_qa_dev1.conf
```

For Intel® Communications Chipset 8925 to 8955 Series:

```
# cp $ICP_ROOT/quickassist/shims/zlib/zlib-1.2.8/contrib/qat/config/dh895xcc/
multi_thread_optimized/dh895xcc_qa_dev0.conf
/etc/dh895xcc_qa_dev0.conf
# cp $ICP_ROOT/quickassist/shims/zlib/zlib-1.2.8/contrib/qat/config/dh895xcc/
multi_thread_optimized/dh895xcc_qa_dev0.conf
/etc/dh895xcc_qa_dev1.conf
```

Note: If more than two acceleration devices are present, copy the same configuration to dev2, dev3, etc. If running with the mux layer, both types of configuration files should be updated as above.

Restart the acceleration service to enable the configuration changes:

```
# service qat_service restart
```

2.2.2 Running the Example Applications

2.2.2.1 zpipe Application

zpipe is an application included with the zlib package that highlights proper usage of the zlib `inflate()` and `deflate()` functions. The application is patched to take advantage of Intel® QuickAssist Technology functionality.

The `zpipe` application can be built with the following commands:

```
# cd $ZLIB_ROOT
# make zpipe
```

The application is located at:

```
$ZLIB_ROOT
```

Note: Prior to running the `zpipe` test application, the Intel® QuickAssist Technology Memory Management kernel module (`qat_mem.ko`) must be installed. Refer to step 12 of Section 2.1, “Installing zlib and Applying the Patch” on page 9.

The following example will compress an example file named file.txt:

```
# cd $ZLIB_ROOT
# ls -l > file.txt
# ./zpipe < file.txt > file.txt.z
```

The output/compressed file generated is file.txt.z.

The following example will uncompress the input file, file.txt.z:

```
# ./zpipe -d < file.txt.z > file.txt
```

The output/uncompressed file generated is file.txt.

2.2.2.2 minigzip Application

The `minigzip` application was built when the `make` in `$ZLIB_ROOT` was executed (Step 14 in Section 2.1). The application is located at:

```
$ICP_ROOT/quickassist/shims/zlib/zlib-1.2.8
```

Note: Prior to running the `minigzip` test application, the Intel® QuickAssist Technology Memory Management kernel module (`qat_mem.ko`) must be installed. Refer to step 12 of Section 2.1, "Installing zlib and Applying the Patch" on page 9.

The example shown below will compress an example file named `file.txt`:

```
# cd $ZLIB_ROOT
# ls -l > file.txt
# ./minigzip file.txt
```

The output/compressed file generated is `file.txt.gz`.

The example shown below will uncompress the input file, `file.txt.gz`:

```
# ./minigzip -d file.txt.gz
```

The output/uncompressed file generated is `file.txt`.

2.2.2.3 comptestapp Test Application

A stand-alone test application called `comptestapp` is also included with this release. `comptestapp` uses Corpus files (Calgary, Canterbury, and Silesia). One or more of the Corpus files must be obtained and placed in the `/lib/firmware` directory as described below.

The Silesia corpus files can be obtained from http://www.data-compression.info/Corpora/SilesiaCorpus/index.htm

Build the `comptestapp` test application:

```
# cd $ZLIB_ROOT/contrib/qat/qat_zlib_test/
# make
```

Usage Information:

```
./comptestapp [-t <type>] [-c <count>] [-n <count>] [-nc <count>] [-k <size>]
[-o <corpus>] [-u] [-af] [-dc] [-dd] [-f <filepath>] [-l <compressionlevel>]
[-ddb] [-dib] [-s <streamtype>] [-w <windowsize>] [-pc] [-pi <interval>] [-v]
[-h]
Where:
    -t specifies the test type to run (see below)
    -c specifies the test iteration count
    -n specifies the number of threads to run
    -nc specifies the number of CPU cores
    -k specifies the chunk size in bytes
    -o specifies the corpus to use for the tests (see below)
```

```
        -u display cpu usage per core
        -af enables core affinity
        -dc disables the QAT Engine for Compression
        -dd disables the QAT Engine for Decompression
        -f specifies the filepath for a corpus test
        -l specifies the compression level
        -ddb disables internal buffering for shim deflate
        -dib disables internal buffering for shim inflate
        -s specifies the type of deflate stream (see below)
        -w specifies the deflate window size (see below)
        -pc allow partial chunks
        -pi specifies the polling interval in nanoseconds
        -v enable verification of data (use with -c 1)
        -h print this usage

    and where the -t test type is:

        1  = Corpus Compression
        2  = Corpus Decompression

    and where the -o corpus is:

        0  = Custom (customfile.bin)
        1  = Canterbury Corpus
        2  = Calgary Corpus
        3  = Silesia Corpus

    and where the -s streamtype is:

        0  = Raw Deflate Stream
        1  = Zlib Format Deflate Stream
        2  = Gzip Format Deflate Stream

    and where the -w windowsize is:

        5  = 8KB Deflate Window Size
        7  = 32KB Deflate Window Size
```

Performance is affected by the choice of options above. The following guidance should help select settings that provide good performance:

For -c the test iterations, you should pick a figure in the thousands, preferably one that is divisible by the number of threads. This allows a good average throughput.

For -n number of threads, you should pick a figure that will keep the accelerator busy but not cause too much contention. Usually figures between 16 and 128 work well, with figures around 20-40 being optimum.

For -nc number of cores, there is no need to restrain the number of cores unless that is specifically the performance you want to measure.

For -k chunksize, you should only need to set this parameter if you disable the shim from buffering up data before submitting to the accelerator using the -ddb or -dib flags. In those cases you should set this parameter fairly high as larger chunksizes will result in less engine requests. Recommended chunksize to show best performance will usually be 32768 bytes or 65536 bytes.

For -l compression level, choosing a lower compression level will increase performance. To show best performance, it is recommended to run with compression level 1.

For -pi polling interval, the default polling interval is 100,000ns. Decreasing this figure will increase CPU usage (and cause more contention) but allow messages to be processed quicker. Increasing the figure has the opposite effect. To show best performance it has been found that a figure between 100,000 and 400,000 gives the best results. There is little benefit to be gained in reducing the polling interval below 100,000ns.

It is not recommended any of the other options be changed or specified when running for best performance.

A good example test to run is:

```
# ./comptestapp -t 1 -c 16384 -n 32 -l 1
```

A test summary is presented when the `comptestapp` test completes. For Intel® Communications Chipset 8900 to 8920 Series fitted with one device, the test summary will look similar to the following:

```
# ./comptestapp -t 1 -c 16384 -n 32 -l 1
<test logging>
Elapsed time   = 52583.293 msec
Operations     = 16384
Time per op    = 3209.429 usec (311 ops/sec)
Elapsed cycles = 136716776365
Throughput     = 8092.39 (Mbps)
```

For the Intel® Communications Chipset 8925 to 8955 Series fitted with one device, the test summary will look similar to the following:

```
# ./comptestapp -t 1 -c 16384 -n 32 -l 1
<test logging>
Elapsed time   = 18739.580 msec
Operations     = 16384
Time per op    = 1143.773 usec (874 ops/sec)
Elapsed cycles = 24361436992
Throughput     = 22707.27 (Mbps)
```

For more information on tuning, please see the *Intel® QuickAssist Technology Performance Optimization Guide* (document number 330687), the *Intel® Communications Chipset 89xx Series Performance Brief* (document number 470252), and the *Intel® Communications Chipset 8950 Performance Brief* (document number 527069).

Note: If the test platform has more than one Intel® QuickAssist Technology accelerator, it is strongly recommended that they have the exact same SKU and configuration. Otherwise, the performance may be limited by the slowest SKU or configuration, due to the implementation of the test application.

Note: Performance may vary depending on the platform configuration, including the version of the patch and the specific product number of the acceleration device.

2.2.3 Verifying Hardware Acceleration

If the performance of the test applications is lower than expected, verify that hardware acceleration is enabled. Check that the hardware has processed requests and responses with the following commands.

For Intel® Communications Chipset 8900 to 8920 Series:

```
# cat /proc/icp_dh89xxcc_dev0/qat0
```

For Intel® Communications Chipset 8925 to 8955 Series:

```
# cat /proc/icp_dh895xcc_dev0/qat
```

2.2.4 Library Linking Instructions

To link an application to the shared library version of accelerated zlib will require linking with the following additional libraries:

```
-ldl -lrt -lpthread
```

To link an application to the static library version of accelerated zlib will require linking with the following additional libraries:

```
-L$(ICP_BUILD_OUTPUT) -licp_qa_al -ladf_proxy -losal -
lcrypto -ldl -lrt -lpthread
```

§ §