INTEL® IN-MEMORY ANALYTICS ACCELERATOR ARCHITECTURE SPECIFICATION

Document Number: 350295-001US
Revision: 1.0
February 2022

Notices & Disclaimers

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

No product or component can be absolutely secure.

Your costs and results may vary.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

All product plans and roadmaps are subject to change without notice.

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.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

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.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that a) you may publish an unmodified copy and b) code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), https://opensource.org/licenses/0BSD. You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

Table of Contents

1 Introduction 9
1.1 Audience 9
1.2 References 10
2 Overview 11
2.1 Data Analytics Features 11
3 Intel Analytics Accelerator Architecture 13
3.1 Operations Overview 13
3.2 Analytics Engine Configuration and State 13
3.3 Decompression 14
3.3.1 Verification 14
3.3.2 Index Generation 15
3.4 Compression 15
3.4.1 Statistics Mode Output 15
3.4.2 Compression Output Overflow 16
3.4.3 Compression Indexing 16
3.5 Checksum Calculations 16
3.6 Filter Functions 17
3.6.1 Parser 17
3.6.2 Packed Array 17
3.6.3 Parquet RLE 17
3.6.4 Output Modification 18
3.6.5 Modification When Output is Normally a Bit Vector 18
3.6.6 Modification When Output is Normally an Array 18
3.6.7 Aggregation 18
3.7 Zero Compression 19
3.7.1 DWORD Zero Compression Format 19
3.7.2 WORD Zero Compression Format 19
3.8 Operation Types 19
3.8.1 Decompress 20
3.8.2 Compress 20
3.8.3 CRC-64 20
3.8.4 Zdecompress32 and Zdecompress16 21
3.8.5 Zcompress32 and Zcompress16 22
3.8.6 Scan 22
3.8.7 Set Membership 22

3.8.8 Extract 22
3.8.9 Select 23
3.8.10 RLE Burst 23
3.8.11 Find Unique 24
3.8.12 Expand 24
4 Error Handling 25
4.1 Descriptor Checks 25
4.2 Descriptor Reserved Field Checking 25
4.3 AECS Checks 28
4.3.1 Compress AECS Checks 28
4.3.2 Decompress/Filter AECS Checks 28
4.4 Error Codes 28
4.4.1 Operation Status Codes 28
4.4.2 Error Code 29
5 Software Architecture 33
5.1 Intel® Query Processing Library 33
6 Structure Formats 35
6.1 Descriptor 35
6.1.1 Trusted Fields 35
6.1.2 Operation 36
6.1.3 Operation Flags 36
6.1.4 Decompression Flags 40
6.1.5 Compression Flags 41
6.1.6 Filter Flags 42
6.1.7 Completion Record Address 43
6.1.8 Source 1 Address 44
6.1.9 Destination Address 44
6.1.10 Source 1 Transfer Size 44
6.1.11 Completion Interrupt Handle 44
6.1.12 Source 2 Address 45
6.1.13 Maximum Destination Size 45
6.1.14 Source 2 Transfer Size 45
6.1.15 Number of Input Elements 45
6.2 Completion Record 47
6.2.1 Status 47
6.2.2 Error Code 47
6.2.3 Bytes Completed 47
6.2.4 Fault Address 48

6.2.5 Invalid Flags 48
6.2.6 Output Size 48
6.2.7 Output Bits 48
6.2.8 XOR Checksum 49
6.2.9 CRC 49
6.2.10 Aggregates 49
6.3 CRC-64 Descriptor and Completion Record 50
6.3.1 CRC Flags 50
6.3.2 CRC Polynomial 51
6.4 Analytics Engine Configuration and State 52
6.4.1 AECS Format for Decompress and Filter 52
6.4.2 AECS Format for Compress 55
7 Summary of Differences from Intel® DSA 57
7.1 General Differences 57
7.2 Configuration and Control Register Differences 58
7.2.1 General Capabilities Register (GENCAP) 58
7.3 PCI Express Configuration Register Differences 58
7.3.1 Device ID (DID) 58
7.3.2 Outstanding Page Request Capacity (PRSREQCAP) 59

List of Figures

Figure 2-1: Intel Analytics Accelerator 11
Figure 6-1: Descriptor Format 35
Figure 6-2: Completion Record Format 47
Figure 6-3: CRC-64 Descriptor Format 50
Figure 6-4 : CRC-64 Completion Record Format 50
Figure 6-5: AECS Format for Decompress and Filter 52
Figure 6-6 AECS Format for Compress 55

List of tables

Table 1-1: References 10
Table 3-1 : AECS Sizes for Various Operations 14
Table 3-2 : Examples of CRC64 Parameters 21
Table 3-3 : Example of RLE Burst Operation With Two Equivalent Primary Inputs 24
Table 4-1: Operation-Specific Flags and Allowed Fields 27
Table 4-2: Conditional Reserved Field Checking 27
Table 4-3 : Operation Types with Required (must be 1) Flags 27
Table 4-4: Operation status codes 29
Table 4-5: Error Codes 31
Table 6-1: Descriptor Trusted Fields 35
Table 6-2: Operation types 36
Table 6-3: Descriptor Flags 39
Table 6-4: Decompression Flags 41
Table 6-5: Compression Flags 42
Table 6-6: Filter Flags 43
Table 6-7: Completion record Status field 47
Table 6-8: Completion record Aggregates fields 49
Table 6-9: AECS fields for Decompress and Filter 53
Table 6-10 Decompress/Analytics Internal State 53
Table 6-11 ALU Field Definitions 54
Table 6-12 AECS Fields for Compress 56

Glossary

Acronym	Term	Description
AECS	Analytics Engine Configuration and State	A data structure used to pass configuration data that did not fit into the descriptor to the accelerator, and to pass state information between descriptor executions when a job consists of multiple descriptors.
DSA	Intel® Data Streaming Accelerator	Intel Accelerator design to accelerate streaming operations such as memory copy and others.
QPL	Intel® Query Processing Library	Intel library to interface between applications and the hardware.

§

1 Introduction

The Intel® In-Memory Analytics Accelerator (Intel® IAA) is a hardware accelerator that provides very high throughput compression and decompression combined with primitive analytic functions.

The Intel® Data Streaming Accelerator (Intel® DSA) is a data mover and transformation accelerator. Intel IAA and Intel DSA share the same hardware/software and programming interface. This document describes the Intel IAA specific functionality and the minor differences in interface from the base Intel DSA specification. One should refer to the Intel Data Streaming Accelerator Architecture specification for details on the common elements.

1.1 Audience

The intended audience for this specification includes hardware engineers and SoC architects to build the hardware implementation, device driver software developers to program the device, virtualization software providers to efficiently enable sharing and virtualization of the device, and application or library developers utilizing accelerator operations.

It is assumed that the reader is already familiar with the Intel Data Streaming Accelerator (Intel DSA) architecture.

1.2 References

Description
Intel® Data Streaming Accelerator Architecture Specification https://software.intel.com/en-us/download/intel-data-streaming-accelerator-preliminary-architecture-specification
Intel® 64 and IA-32 Architectures Software Developer's Manuals https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html
Intel® Architecture Instruction Set Extensions Programming Reference https://software.intel.com/content/www/us/en/develop/download/intel-architecture-instruction-set-extensions-programming-reference.html
Intel® Query Processing Library https://github.com/intel/qpl
PCI Express* Base Specification 4.0 http://www.pcisig.com/specifications/pciexpress
Intel® Virtualization Technology for Directed I/O Specification https://software.intel.com/content/www/us/en/develop/download/intel-virtualization-technology-for-directed-io-architecture-specification.html
Intel® Scalable I/O Virtualization Technical Specification https://software.intel.com/content/www/us/en/develop/download/intel-scalable-io-virtualization-technical-specification.html
Intel® I/O Acceleration Technology https://www.intel.com/content/www/us/en/wireless-network/accel-technology.html
ITU-T recommendation V.42 https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-V.42-200203-I!!PDF-E&type=items
RFC 1951, DEFLATE Compressed Data Format Specification http://www.ietf.org/rfc/rfc1951.txt
RFC 3720, Internet Small Computer Systems Interface http://www.ietf.org/rfc/rfc3720.txt

Table 1-1: References

§

2 Overview

The Intel In-Memory Analytics Accelerator (Intel IAA) is a hardware accelerator that provides very high throughput compression and decompression combined with analytic primitive functions. The analytic functions are commonly used for filtering data during analytic query processing. It primarily targets applications such as big-data and in-memory analytic databases, as well as application-transparent usages such as memory page compression. Other operations such as data integrity functions (e.g., CRC64) are also supported. The device supports light-weight compression schemes such as zero-compression as well as heavier formats such as Huffman encoding and Deflate. For the Deflate format, it supports indexing the compressed stream for efficient random access.

2.1 Data Analytics Features

The accelerator contains two main functional blocks: Compression and Analytics. The Analytics pipe contains two sub-blocks: Decompress and Filter. These functions are tied together, so that each analytics operation can perform decompress-only, filter-only, or decompress-and-filter processing, as illustrated in Figure 2-1. Alternatively, one can compress the input.

The accelerator allows storing columnar databases in compressed form, decreasing memory footprint. In addition to increased effective memory capacity, this also reduces memory bandwidth by performing the filter function used for database queries “on the fly”, thereby avoiding use of memory bandwidth for uncompressed raw data transfer.

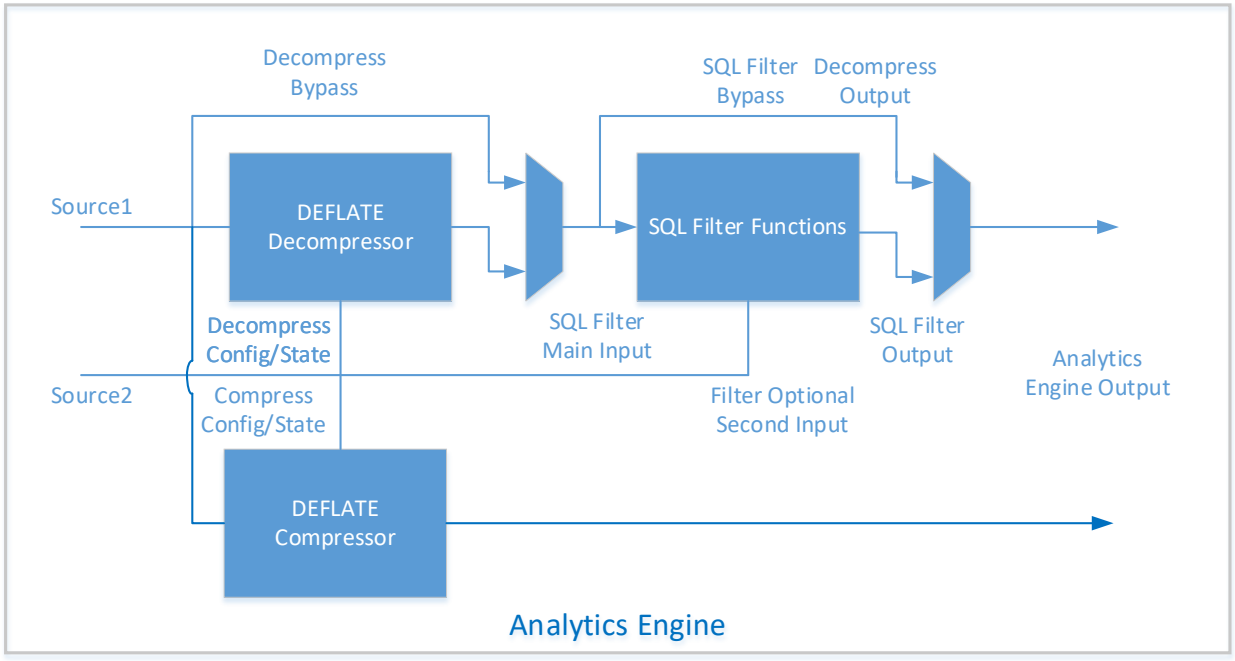

Figure 2-1: Intel Analytics Accelerator

The device supports decompression compatible with the Deflate compression standard described in RFC 1951. The uncompressed data may be written directly to memory or passed to the input of the filter function. Decompression is supported for Deflate streams where the size of the history buffer is no more than 4096.

It also supports Deflate compression, along with the calculation of arbitrary CRCs and two varieties of zero compression/decompression.

The SQL filter function block takes one or two input streams, a primary input, and an optional secondary input. The primary input may be read from memory or received from the decompression block. The secondary input, if used, is always read from memory. The data streams logically contain an array of unsigned values, but they may be formatted in any of several ways, e.g., as a packed array. If the bit-width of the values is 1, the stream will be referenced as a “bit-vector”, otherwise it will be referenced as an “array”.

The output of the filter function may be either an array or a bit vector depending on the function.

In addition to generating output data, the device computes a 32-bit CRC of the uncompressed data (either the result of decompression or the direct input to the filter function), the XOR checksum of this data, and several “aggregates” of the output data. The CRC, XOR checksum, and aggregates are written to the completion record.

§

3 Intel Analytics Accelerator Architecture

3.1 Operations Overview

The following data operations are supported by the accelerator. The following sections give more details on these operations.

Type	Operation	Description
Decompress	Decompress	Decompress input data.
Compress	Compress	Compress input data.
Filter	Scan	Compute a bit-mask of which entries satisfy a condition.
	Set Membership	Compute a bit-mask of which entries belong to a specified set.
	Extract	Return entries as specified by a range of entry indices.
	Select	Return entries as specified by a bit-mask.
	RLE Burst	Expand entries that have been RLE-encoded.
	Find Unique	Return a bit-mask indicating which values are present in the input.
	Expand	Insert zeros as specified by a bit-mask.
Zero Compress	Zcompress16	Compress by removing 16-bit words containing zeros.
	Zcompress32	Compress by removing 32-bit words containing zeros.
	Zdecompress16	Decompress data that was compressed by Zcompress16.
	Zdecompress32	Decompress data that was compressed by Zcompress32.
CRC	CRC-64	Compute an arbitrary CRC up to 64-bits in size.
Move	Memory Move	Copy data from input to output.

3.2 Analytics Engine Configuration and State

The analytics engine configuration and state structure (AECS) contain configuration information that is used to control the behavior of the decompressor and the filter functions. Details of this structure are in section 6.4. In addition to configuration information, the AECS may contain internal state of the analytics engine. The state information can be used to initialize the engine to a known state and to propagate state information from one operation to another. For each operation, the AECS may be read or written or both, depending on flags in the descriptor, as described in section 6.1.3.

When Source 2 Address and Source 2 Size are being used to read and/or write the AECS, then the actual memory being referenced will be twice the specified size. The read will occur from one half of the area, and the write will occur to the other half. In this way, the input data will not get overwritten by the output data, so that in the event of an error, the request can be retried by software. The AECS R/W Toggle Selector bit in the Operations Flags field of the descriptor indicates which half supplies the read data, and which half receives the write data.

In particular, if the AECS address (i.e., Source 2 Address) is “A”, and the AECS size (i.e., Source 2 Transfer Size) is “S”, then in one case the AECS is read from (A) and written to (A+S), and in the other case it is read from (A+S) and written to (A). Note that the total amount of memory accessed would be in general (2S).

Note that in some cases the AECS may be read but not written or written but not read. In either of these cases, the address used for the read or the write is the same as if there was both a read and a write happening. For example, if the AECS was being read but not written, and the AECS R/W Toggle Selector was 1, then the AECS would be read from (A+S) and nothing would be written to (A).

Depending on the operation, some portions of the nominal AECS may not be relevant and do not need to be read/written. Since the AECS is always read starting at the beginning, the only portion that can be omitted is at the end. This implies that the size specified for the AECS (i.e., the Source 2 Transfer Size) will vary depending on the operation. In general, this value will be the size of the relevant data (starting at offset 0) rounded up to the next multiple of 32. So typical values would be:

Operation	AECS Size
Filter	32
Decompress	5376
Compress (With Huffman Table)	1568

Table 3-1 : AECS Sizes for Various Operations

3.3 Decompression

Intel IAA supports decompression compatible with the Deflate compression standard described in RFC 1951. The decompression block reads a compressed stream and an optional AECS and generates the corresponding uncompressed data. The uncompressed data may be written directly to memory or passed to the input of the filter function.

Decompression can be performed on a single buffer, where the entire stream is contained in a single buffer, or on multiple buffers, and the stream spans more than one buffer. In the latter case, a separate descriptor is submitted for each buffer. This is called a job. That is, a job is a series of descriptors that operate on one logical stream. The descriptors in a job are tied together by the use of a common AECS. The AECS written by each descriptor in the job is read by the next descriptor. The AECS stream contains data used to connect the individual descriptors used to process one logical stream. It is typically read on all but the initial descriptor of a job, and it is written on all but the final descriptor.

For operations that write the output of decompression to memory, the output buffer size specified in the descriptor should be large enough to hold the output of the operation. If the output does not fit into the specified output buffer, the decompression operation terminates and reports the amount of the input that was consumed. An additional descriptor must be submitted to process the remaining input data into a new output buffer.

Decompression is supported for Deflate streams where the size of the history buffer is no more than 4 KB. (The default size for Deflate is 32 KB.) Using an input stream with a larger history size results in an error.

3.3.1 Verification

The decompression operation can be used by software to verify that the output generated by compression is correct, i.e., that it can be decompressed back to the original input. This could be done as a normal decompression job, with the output going into a buffer that is then compared against the original input.

A more efficient approach is to suppress the output of the decompressor. In this case, the hardware would write no output data, but it would still calculate the CRC of the decompressed data. This can then be compared against the CRC computed from the input to the compressor.

This avoids the need to have a temporary buffer in which to write the decompressed data, the overhead of the compare operation, and the bandwidth needed to write and read that data.

3.3.2 Index Generation

The generation of indices for the compressed data (cf. Section 3.4.3) is done by the decompressor while it is operating for verification. In this case, the normal decompressed output has to be suppressed. Then when indexing is enabled, the index data is written to the output buffer. Note that the compression must have been done with indexing enabled.

The flush flag cannot be used when indexing is being used, except for a last job (i.e., when Write_AECS is "never").

3.4 Compression

Intel IAA supports compression compatible with the Deflate compression standard described in RFC 1951. The compression unit can operate in two modes: Huffman-mode or Statistics-mode.

In Huffman-mode, it will read a stream of input bytes, generate a stream of literals and matches, encode them using a Huffman table read from the AECS, and write those Huffman codes into the output buffer.

In Statistics-mode, rather than writing the Huffman Tokens to the output buffer, it will instead compute a histogram of how many times each Huffman code appears. At the end of processing, the histogram table is written to the output buffer.

To generate a dynamic Deflate block, the software should do one pass in statistics mode, use the statistics to generate a set of Huffman Tables optimized for those statistics, and then do a second pass (with the same input data) in Huffman-mode.

The hardware will optionally add an EOB (End of Block) token to the output or add an EOB and a zero-length Stored Block to the output. The block header, however, should be added to the output accumulator in the AECS by software before submitting the descriptor.

3.4.1 Statistics Mode Output

The format of the histogram table output in Statistics Mode is as a table of 318 32-bit words:

Byte Offset	Description
0	LitLen[0] count
…	…
1140	LitLen[285] count
1144	Reserved
1148	Reserved
1152	Distance[0] count
…	…
1268	Distance[29] count

These give the number of times each of 286 Literal/Length Tokens appeared, and the number of times the 30 Distance Tokens appeared. Note that while each count occupies a 32-bit field, the actual counts are 19-bits wide. If 2^{19} or more of a given token appears, the count saturates at (2^{19}-1).

3.4.2 Compression Output Overflow

For compression, "output overflow" is a non-recoverable error, and the AECS is not written.

The output buffer should be sized slightly larger than the input buffer, such that the input buffer could be encoded as a Deflate stored-block, written to the output buffer, and fit. In that case, if the compression operation actually results in data expansion such that the compressed data would not fit into the output buffer, the software (library or application) should ignore any partial results that the compressor generated and add the current input to the output stream as a stored block. This would result in a better compression ratio than keeping the "compressed" data.

3.4.3 Compression Indexing

The compression logic also supports "Indexing". When this is enabled, it also defines a "mini-block size". The meaning of this is that no match will cross a mini-block boundary, and no match will reference data in a different mini-block. This will allow a decompressor to start decompression at a mini-block boundary at the cost of a slightly reduced compression ratio.

If Compression Indexing is enabled, then the input buffer must be a multiple of the mini-block size, except for the last descriptor of a job.

For indexing to work properly, the application must know the block structure of the output. This means that either the compressed output must fit within the provided output buffer (i.e., no "output overflow") or the input buffer must be smaller than 64kB, so that it will fit into a single stored block.

3.5 Checksum Calculations

The accelerator generates a 32-bit CRC of the uncompressed data (either the result of decompression or the direct input to the filter function). More particularly, the checksums in the Analytics pipe are computed on the data after any Decompress and before any Filter. For Compress functions, the checksums are computed on the input.

The user can select either 32-bit CRC as defined in ITU-T recommendation V.42 and in RFC 3720. It also computes the XOR checksum of the uncompressed data, treated as 16-bit words. If there are an odd number of bytes, the final byte is zero-extended to 16 bits.

The initial values of the CRC and XOR checksum are read from the AECS, for operations where the AECS is read; otherwise, initial values of 0 are used. The final values of the checksums are written to the completion record. They are also written to the AECS, for any operation where the AECS is written. The latter allows the values to be linked across the descriptors in a job, while the former allows the software to get the values even when the AECS is not written.

3.6 Filter Functions

The filter functions take one or two inputs, a primary input, and an optional secondary input. The primary input may be read from memory or received from the output of decompression. The primary input is parsed as described in 3.6.1. The output of the parser is an array of unsigned integers.

If the secondary input is used, depending on the operation type, it may be a bit vector or an array of packed unsigned integers. It can be packed in either little-endian format (starting at bit 0 of each byte) or big-endian format (starting at bit 7). When the secondary input is used, the operation cannot also use the AECS. Thus, any operation that uses the secondary input uses default values for any configuration information that would have been read from the AECS.

The output of the filter function may be either an array or a bit vector depending on the function.

For filter operations, the output buffer size specified in the descriptor must be large enough to hold the entire output of the operation. If the output does not fit into the specified output buffer, the operation fails with an unknown amount of the input processed. In this case, the software needs to resubmit the descriptor with a larger output buffer.

3.6.1 Parser

One of the following parsers may be selected to process the primary input to the filter function. The parser reads a byte stream and outputs a series of unsigned integers.

3.6.2 Packed Array

This is the standard parser. The input is a packed array of unsigned integers with a specified bit width. (The bit width need not be a multiple of the size of a byte.) The data can be packed in little-endian format (starting at bit 0 of each byte) or big-endian format (starting at bit 7).

3.6.3 Parquet RLE

The input is in the Parquet RLE format. The first byte of the data stream gives the bit width. This is followed by the encoded data. The bit-width cannot exceed 32-bits.

The format is:

parquet-rle: <bit-width> <encoded-data>
bit-width := bit-width of data stored as one byte
encoded-data := <run>*
run := <bit-packed-run> | <rle-run>
bit-packed-run := <bit-packed-header> <bit-packed-values>
bit-packed-header := varint-encode(<bit-pack-count> << 1 | 1)
// we always bit-pack a multiple of 8 values at a time, so we only store the number of values / 8
bit-pack-count := (number of values in this run) / 8
bit-packed-values := data stored as a packed array of bit-width values
rle-run := <rle-header> <repeated-value>
rle-header := varint-encode((number of times repeated) << 1)
repeated-value := value that is repeated, using a fixed-width of round-up-to-next-byte(bit-width)

3.6.4 Output Modification

Output Modification is an SQL filter feature that allows optionally generating an alternative representation of the result of the query being performed. In general, a SQL filter function results in two forms of outputs: 1) a bit vector (where the output bit-width is 1) or 2) an array of elements (where the output bit-width is greater than 1). For example, functions that perform a set-membership query, i.e., "is an element a member of a given set?", generate a bit vector, where each bit represents membership in the set. Functions that extract elements from an input array result in an output that could be either a bit-vector or an array depending on the bit-width of the input.

3.6.5 Modification When Output is Normally a Bit Vector

If the output of that function is normally a bit vector, the output can be modified in the following ways.

First, the bit vector can be optionally inverted (i.e., each bit is flipped).

Secondly, the output can be modified to consist of an array, where the array elements are the indices of the "1" bits of the bit vector. This can be used when the output bit vector is expected to be sparse in nature. The index of the first element (bit 0 of the bit vector) can be set to an arbitrary value instead of the default start index of 0. If any element is too large for the specified output width, the operation stops and reports an error.

3.6.6 Modification When Output is Normally an Array

If the output of a function is normally an array of elements, then the bit width of the output elements is normally the same as the input bit width; i.e., the output is packed. When the output modification feature is enabled, output bit width can be adjusted to 8, 16, or 32 (with the high order bits padded with zeroes). This unpacks the output array into a desired word size. Using this feature makes the output array larger, but it makes it easier for software to process the data. The specified output bit width must be no smaller than the input bit width.

If the output bit width is 1, the output is treated as a bit vector, and the output modifications described in section 3.6.5 apply.

3.6.7 Aggregation

In addition to generating the output data, the accelerator also computes several "aggregates" of the data. The type of aggregation depends on the type of output. In particular, it depends on the nominal, pre-modified output width: whether it is 1-bit wide or wider.

If the "pre-modification" output is a bit vector or an array output whose bit width is 1, then the following data are accumulated:

- Population count (the number of 1 bits)
- First (the index of the first 1 bit)
- Last (the index of the last 1 bit)

This data can be used to determine the sparsity of the output. If the output is sparse, software can use it to determine where to start and end processing, so that it doesn't need to process the 0 bits at the start or end of the vector.

If the "pre-modification" output is an array whose bit-width is greater than 1, then the following data are accumulated:

- Sum (the sum mod 2^{32} of the output values)
- Minimum value
- Maximum value

Note that the population count is actually a special case of "sum".

3.7 Zero Compression

The accelerator also supports a lightweight compression scheme called "zero-compression". This can operate on 32-bit words or on 16-bit words. For DWORD compression/decompression, the input must be an integral number of DWORDs. For word compression/decompression, the input must be an integral number of words.

Conceptually, the zero-compress operation is equivalent to doing a "scan neq zero" followed by a "select", except that the tags and data are interleaved. Similarly, the zero-decompress operation is equivalent to doing an "expand", except with interleaved tags and data

3.7.1 DWORD Zero Compression Format

The data is compressed 32-DWORDs at a time. The output consists of a 32-bit tag DWORD, followed by 0 through 32 data DWORDs. The bits in the tag DWORD (starting at little-endian bit 0) indicate whether the corresponding DWORD in the uncompressed data was zero or not. A tag bit of "0" indicates that the original data DWORD was zero, and a "1" indicates that it was non-zero.

The tag DWORD is followed in the compressed data stream by one data DWORD for each 1-bit in the tag (starting at bit-0). So, the number of data DWORDs following each tag DWORD is the number of 1-bits in the tag.

The exception to this is the final record. DWORDs that would appear after the end of the actual data are represented by 1-bits in the tag and no corresponding data DWORDs. For example, if the last tag had the value 0xFFFFFFF9, and it was followed by 3 DWORDS (A, B, and C), then the decompressed output would be: A, 0, 0, B, C, and then the output stream would end.

3.7.2 WORD Zero Compression Format

This is similar to the DWORD format, except that it operates on 16-bit words. The data is compressed 64-words at a time, so the tag consists of 64-bits or 4 words. Word-0 contains tag bits 15-0, word-1 contains tag bits 31-16, etc.

3.8 Operation Types

The operations No-op and Drain are the same as in Intel DSA. The Memory Move operation is the same except that use with overlapping buffers is not supported.

3.8.1 Decompress

The Decompress operation decompresses the input and writes the decompressed data to memory. The Source 1 Address and Source 1 Transfer Size specify the location of the compressed input data. The Destination Address and Maximum Destination Size specify the location of the decompressed output data. The Source 2 Address and Source 2 Transfer Size optionally specify the AECS. The Read Source 2 and Write Source 2 fields indicate the usage of the AECS (read, written, neither, or both). Decompression Flags controls aspects of the decompression operation. The Filter Flags field is reserved. The "Enable Decompression" flag must be set.

If the Status of the operation is Output buffer overflow, the decompression job can be resumed by submitting a follow-on descriptor with a new buffer to contain the remaining decompressed output. The Write Source 2 flag should be 2 for the final (or only) descriptor in a decompression job, to ensure that the state of the decompressor can be saved in the AECS in case of output buffer overflow. The Write Source 2 flag should be 1 for descriptors before the final descriptor in a multiple-descriptor job. The Read Source 2 flag should be 1 for all but the first descriptor in a multiple-descriptor job.

The output may be suppressed (for verification purposes) or replaced with index output (see Section 3.3.2). If the Enable Indexing flag is set, then the Suppress Output flag must also be set.

3.8.2 Compress

The Compress operation compresses the input and writes the compressed data to memory. The Source 1 Address and Source 1 Transfer Size specify the location of the input data. The Destination Address and Maximum Destination Size specify the location of the compressed output data. The Source 2 Address and Source 2 Transfer Size optionally specify the AECS. The Read Source 2 and Write Source 2 fields indicate the usage of the AECS (read, written, neither, or both). The Compression/Decompression Flags control aspects of the compression operation. The Filter Flags field is reserved.

If the compressed output does not fit into the output buffer, the operation fails with an error.

In one usage, if the user wants to ensure that the input can be compressed into a legal Deflate stream, they can size the output buffer to be slightly larger than the input buffer, in particular large enough to contain the uncompressed input as a Deflate "stored block". Then if the compressed output does not fit, the hardware will return an error, and the software (e.g., Intel QPL library) can throw away the partial results generated by the hardware and write a stored block into the output buffer.

In another usage, the application may want to store the data in a compressed form if that form is smaller than the uncompressed data, or store the data uncompressed if the compressed form is larger. In this case, the application could provide an output buffer the same size as the input buffer. Then if the hardware returns an error that the compressed data does not fit, the application can store the data uncompressed.

3.8.3 CRC-64

The CRC-64 operation computes an arbitrary CRC up to 64-bits in width.

The CRC operation uses a modified descriptor, which is described in Section 6.3.

The CRC Bit Order flag indicates whether bit-0 in each data byte is the least-significant or the most-significant bit. Having bit-0 be least-significant corresponds to the "normal form" of the data, whereas having

bit-0 be most-significant corresponds to the "bit-reversed form" of the data. This field also impacts the byte order of the CRC output. If bit-0 is least-significant, then the least significant bit of the CRC is bit-0 of byte-0. If bit-0 is most significant, then the least significant bit of the CRC is bit-7 of byte-7, or bit 63 of the CRC.

The CRC is essentially the residue (remainder) after polynomial division. The "initial value" of the CRC is essentially a constant that is XORed with the initial data bytes. This constant has the same size in bits as the polynomial. In some CRCs, this initial value is zero. In others, it is all 1's. This is determined by the "invert CRC" flag bit. If this flag bit is 0, then the initial value is 0. If the flag bit is set, then the initial value is the "bitwise inverse of 0" or all 1's. As described below, this feature can be used to compute the CRC for polynomials smaller than 64-bits. Because of this, when the "invert CRC" flag is set, the initial value will only have 1-bits from the least-significant 1-bit in the polynomial to the most significant bit. E.g., if the polynomial represents a 32-bit CRC, the initial value will only have 32 1-bits.

Additionally, if the invert CRC flag is set, the final residue is XORed with the initial value before being returned.

The CRC Polynomial field defines the CRC polynomial in normal (not bit-reversed) form, regardless of the state of the CRC Bit Order flag. In the polynomial definition, bit-63 is always most significant.

Although this operation is designed to generate 64-bit CRCs, it can also be used to generate smaller arbitrary CRCs. In that case, the polynomial is placed in the most-significant portion of the CRC Polynomial field (i.e., starting at bit 63), and the results are found in the most-significant portion of the CRC64 field in the completion record, whose location *does* depend on the value of the CRC Bit Order flag.

Here are some well-known CRCs and the programming needed to generate them:

CRC	Polynomial	Bit Order	Invert CRC	Output
CRC32 (gzip)	04C11DB700000000	1	1	00000000XXXXXXXX
CRC32 (wimax)	04C11DB700000000	0	1	XXXXXXXX00000000
CRC32 (iSCSI)	1EDC6F4100000000	1	1	00000000XXXXXXXX
T10DIF	8BB7000000000000	0	0	XXXX000000000000
CRC-16-CCITT	1021000000000000	1	1	000000000000XXXX

Table 3-2 : Examples of CRC64 Parameters

3.8.4 Zdecompress32 and Zdecompress16

These operations decompress data that has been zero-compressed (using DWORDs or words). See Section 3.7 for a description of Zero Compression.

The Source 1 Address and Source 1 Transfer Size specify the location of the input data. The Destination Address and Maximum Destination Size specify the location of the decompressed output.

The following fields of the descriptor are reserved: Source 2 Address, Source 2 Transfer Size, Decompression Flags, and Filter Flags.

For these operations, the CRC and checksum are computed on the output.

3.8.5 Zcompress32 and Zcompress16

These operations zero-compress the DWORDs or words of the input and write the compressed data to memory. See Section 3.7 for a description of Zero Compression.

The Source 1 Address and Source 1 Transfer Size specify the location of the input data. The Destination Address and Maximum Destination Size specify the location of the compressed output.

The following fields of the descriptor are reserved: Source 2 Address, Source 2 Transfer Size, Decompression Flags, and Filter Flags.

For these operations, the CRC and checksum are calculated on the input.

3.8.6 Scan

The Scan operation determines whether each element in the input data stream is in the inclusive range defined by the configuration variables Low Filter Param and High Filter Param (i.e., if (Low Filter Param ≤ element value ≤ High Filter Param)). The output is a bit vector where each 1 indicates that the corresponding input element is in the range.

The output may be modified by inverting each bit and/or by converting to an array of indices.

By selecting suitable values for the parameters and the Invert Output Bits flag, any of the following filter functions may be realized: =, ≠, <, ≤, ≥, >, within a range, and outside a range.

3.8.7 Set Membership

The Set Membership operation determines whether each element in the primary input is contained in the set specified by the secondary input. The output is a bit vector where each 1 indicates that the corresponding input element is a member of the set.

Some bits of each input element may be ignored by setting the Drop Low Bits and Drop High Bits options. The N bits that remain comprise the value to be tested. N is the bit width of the source data minus the number of dropped low and/or high bits.

The set is specified by the secondary input as a bit vector of length 2^N bits. For each bit *i* in the bit vector, the value 1 indicates that the value *i* is a member of the set.

The maximum supported value of N is dependent on the Enable Decompression flag. If Enable Decompression is 1, the maximum value of N is given by the Maximum Decompression Set Size field in GENCAP. If Enable Decompression is 0, the maximum value is given by the Maximum Set Size field in GENCAP.

The output may be modified by inverting each bit and/or by converting to an array of indices.

3.8.8 Extract

The Extract operation returns the elements in the input data stream whose indices fall within the range defined by the configuration variables Low Filter Param and High Filter Param. The indices of the input values are assigned sequentially starting with 0. The output is an array of the input values whose indices fall within the range.

By default, the output bit width is the same as the input bit width. The output may be modified as described in Section 3.6.4.

If Low Filter Param is 0 and High Filter Param is at least the number of elements in the input, then all elements are extracted. With output modification, this can be used to unpack a packed array to a desired word size (byte, word, or DWORD).

3.8.9 Select

The Select operation returns the elements in the primary input whose indices correspond to 1-bits in the secondary input. The indices of the input values are assigned sequentially starting with 0. The output is an array of the input values selected by the bit vector.

The secondary input is a bit vector with at least as many bits as the number of elements in the input.

By default, the output bit width is the same as the input bit width. The output may be modified as described in Section 3.6.4.

3.8.10 RLE Burst

The RLE Burst operation replicates each element in the secondary input a number of times based on the corresponding element in the primary input.

The bit width of the primary input must be 8, 16, or 32. The behavior of the operation depends on the bit width of the primary input, as described below.

The secondary input is a packed array of unsigned integers of any width from 1 to 32 bits. If the bit width of the primary input is 8 or 16 bits, the secondary input has the same number of elements as the primary input. If the bit width of the primary input is 32 bits, the secondary input has one fewer elements than the primary input.

By default, the output bit width is the same as the secondary input bit width. The output may be modified by zero-extending each output value to 8, 16, or 32 bits.

If the bit width of the primary input is 8 or 16, each element specifies the number of times to replicate the corresponding element in the secondary input. If the value is 0, the corresponding element is dropped.

If the bit width of the primary input is 32, each element of the primary input specifies the cumulative number of elements in the output to that point. Thus, the repetition count for each element is the difference between the element and the next. In this case, the first element should always be 0. The number of times each bit is replicated is limited to the range 0–65,535 inclusive. For example, with the first element equal to 0 and the second element equal to 3, the first element of the secondary input is replicated three times in the output. As another example, the two primary inputs shown in Table 3-3 are equivalent.

Secondary Input (1 bit)	Primary input		Output
	Length (8 bits)	Cumulative Length (32 bits)	
1	2	0	11
0	4	2	110000
1	3	6	110000111
0	3	9	110000111000
1	2	12	11000011100011
		14	

Table 3-3 : Example of RLE Burst Operation With Two Equivalent Primary Inputs

3.8.11 Find Unique

The Find Unique operation generates a bit vector where each 1 value indicates that the corresponding value was present in the input.

Some bits of each input element may be ignored by setting the Drop Low Bits and Drop High Bits options. The N bits that remain comprise the value to be tested. N is the bit width of the source data minus the number of dropped low and/or high bits.

The output is a bit vector of length 2^N bits. For each bit *i* in the bit vector, the value 1 indicates that the value *i* was present in the input.

The maximum supported value of N is dependent on the Enable Decompression flag. If Enable Decompression is 1, the maximum value of N is given by the Maximum Decompression Set Size field in GENCAP. If Enable Decompression is 0, the maximum value is given by the Maximum Set Size field in GENCAP.

The output may be modified by inverting each bit and/or by converting to an array. If the output is modified to an array, the values in the array comprise a list of the unique values in the input, zero-extended to 8, 16, or 32 bits.

3.8.12 Expand

The Expand operation generates an array in which the elements in the primary input are placed according to 1 bits in the secondary input. The secondary input is a bit vector. The number of elements in the output is the same as the length of the secondary input. For each bit in the secondary input that is 1, the corresponding value in the output is the next sequential value taken from the primary input. For each bit in the secondary input that is 0, the corresponding value in the output is 0.

For this operation, the descriptor field named Number of Input Elements contains the number of bits in the secondary input, rather than the primary input. The number of elements in the primary input is the same as the number of 1 bits in the secondary input.

By default, the output bit width is the same as the input bit width. The output may be modified as described in Section 3.6.4.

§

4 Error Handling

4.1 Descriptor Checks

For the set of features and operations common to both Intel IAA and Intel DSA, the device performs the checks on each descriptor as described in the Intel DSA Architecture specification. Some additional checks/clarification on checks are that an error will be generated when any of the following are violated:

- No unsupported flags in any of the flag fields are set. This includes flags that are reserved for use with certain operations or that are disabled in the configuration. Flags fields include Operation Flags, Decompression Flags, Compression Flags, and Filter Flags. See Table 4-1 and Table 4-2 for details.
- Required flags in the Flags field are set. For example, the Request Completion Record flag must be 1 in a descriptor for any operation other than No-op, Drain, and Memory Move. See Table 4-3 for details.
- The Source 1 Transfer Size, Source 2 Transfer Size, and Maximum Destination Size (if applicable for the descriptor type) are not greater than the value specified by the WQ Maximum Transfer Size field in the WQ Config register and are non-zero if required by the operation.
- The destination buffer does not overlap the source 1 buffer.
- If Read Source 2 or Write Source 2 is non-zero, the source 2 buffer does not overlap the source 1 buffer or the destination buffer.

4.2 Descriptor Reserved Field Checking

Reserved fields in descriptors fall into three categories: fields that are always reserved; fields that are reserved under some conditions (e.g., based on a capability, configuration field, how the descriptor was submitted, or values of other fields in the descriptor itself); and fields that are reserved based on the operation type. For additional details on descriptor formats, see chapter 6.

Table 4-1 lists the flags and fields that are allowed for each operation type. Flag bits 23:22, 7:6, and 0 are reserved for all operation types. Table 4-2 and Table 4-3 list the differences from Intel DSA for additional conditions under which certain flags and fields are reserved or required. Additional operation-specific reserved fields and flags are described with the respective descriptor details in chapter 6.

		No-op	Drain	Memory Move	Decompress	Compress	CRC-64	Zdecompress32	Zdecompress16	Zcompress32	Zcompress16	Scan	Set Membership	Extract	Select	RLE Burst	Find Unique	Expand
Op Flags	Block On Fault			•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	Comp Rec Addr Valid	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	Req Comp Record	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	Req Comp Interrupt	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

		No-op	Drain	Memory Move	Decompress	Compress	CRC-64	Zdecompress32	Zdecompress16	Zcompress32	Zcompress16	Scan	Set Membership	Extract	Select	RLE Burst	Find Unique	Expand
	Completion Record TC Selector	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	Completion Record Steering Tag Selector	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	Source 1 TC Selector		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	Dest TC Selector		•	•	•	•		•	•	•	•	•	•	•	•	•	•	•
	Dest Steering Tag Selector			•	•	•		•	•	•	•	•	•	•	•	•	•	•
	Cache Control			•	•	•		•	•	•	•	•	•	•	•	•	•	•
	Strict Ordering			•	•	•		•	•	•	•	•	•	•	•	•	•	•
	Dest Readback			•	•	•		•	•	•	•	•	•	•	•	•	•	•
	Read Source 2				•	•						•	•	•	•	•	•	•
	Write Source 2				•	•												
	Source 2 TC Selector				•	•						•	•	•	•	•	•	•
	Source 2 Steering Tag Selector				•	•												
	CRC Select				•	•		•	•	•	•	•	•	•	•	•	•	•
Decompress Flags	Enable Decompression				•							•	•	•	•	•	•	•
	Flush Output				•							•	•	•	•	•	•	•
	Stop on EOB				•							•	•	•	•	•	•	•
	Check for EOB				•							•	•	•	•	•	•	•
	Select Bfinal EOB				•							•	•	•	•	•	•	•
	Decompress Bit Order				•							•	•	•	•	•	•	•
	Ignore End Bits				•							•	•	•	•	•	•	•
	Suppress Output				•							•	•	•	•	•	•	•
	Enable Indexing				•													
Compress Flags	Stats Mode					•												
	Flush Output					•												
	End Processing					•												
	Generate All Literals					•												
	Compress Bit Order					•												
	Enable Indexing					•												
CRC Flags	Invert CRC						•											
	CRC Bit Order						•											
Filter Flags	Source 1 Parser											•	•	•	•	•	•	•
	Source 1 Width											•	•	•	•	•	•	•

			No-op	Drain	Memory Move	Decompress	Compress	CRC-64	Zdecompress32	Zdecompress16	Zcompress32	Zcompress16	Scan	Set Membership	Extract	Select	RLE Burst	Find Unique	Expand
		Source 2 Width															•		
		Source 2 Bit Order												•		•	•		•
		Output Width											•	•	•	•	•	•	•
		Output Bit Order											•	•	•	•	•	•	•
		Invert Output											•	•	•	•	•	•	•
		Drop Low Bits												•				•	
		Drop High Bits												•				•	
Bytes	16–23	Source 1 Address			•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	24–31	Destination Address			•	•	•		•	•	•	•	•	•	•	•	•	•	•
	32–35	Source 1 Size			•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	38–39	Misc Flags				•	•	•					•	•	•	•	•	•	•
	40–47	Source 2 Address				•	•						•	•	•	•	•	•	•
	48–51	Destination Size				•	•		•	•	•	•	•	•	•	•	•	•	•
	52–55	Source 2 Size				•	•						•	•	•	•	•	•	•
	56–63	Filter/CRC-64						•					•	•	•	•	•	•	•

Table 4-1: Operation-Specific Flags and Allowed Fields

Reserved Field	Conditions under which field is reserved
Source 2 Address	Read Source 2 = 0 and Write Source 2 = 0.
Source 2 Size	Read Source 2 = 0 and Write Source 2 = 0.
Source 2 TC Selector	Read Source 2 = 0 and Write Source 2 = 0.
Source 2 Steering Tag Selector	Read Source 2 = 0 and Write Source 2 = 0 or TPH Requester Control Register ST Mode Select = 0.

Table 4-2: Conditional Reserved Field Checking

Table 4-3 gives the list of Intel IAA-specific operation types that require certain flags to be set to 1.

Operation	Required Flags (must be 1)
All operations other than No-op, Drain, and Memory Move	Completion Record Address Valid and Request Completion Record flags must be 1.
Decompress	Suppress Output must be set if Enable Indexing is non-zero.
Decompress	Enable Decompression must be 1.

Table 4-3 : Operation Types with Required (must be 1) Flags

4.3 AECS Checks

When Source-2 is read as AECS (cf. Section 6.4), a number of checks are performed on the data. If any of these are violated, then the operation terminates with an error with no further processing.

The checks performed depend on whether the operation is Compress or Decompress/Filter.

4.3.1 Compress AECS Checks

If any of the following constraints is violated, processing halts with an error:

- Number of Output Accumulator Bits Valid ≤ 64 × 32.

4.3.2 Decompress/Filter AECS Checks

If any of the following constraints is violated, processing halts with an error:

- Input Accumulator Size QW ≤ 64.
- If (Input Accumulator Size QW[i] = 0) then Input Accumulator Size QW[i+1] must be 0.
- If ((Decompress State & 0xD) = 2) (i.e., looking at Stored Block), then Stored Block Bytes Remaining must be non-zero.
- The Output Bits Valid must be a multiple of 8.
- If Indexing is enabled, then (History Buffer Write Pointer{2:0}) must be the same as (Output Bits Valid{5:3}).
- If the History Buffer Write Pointer is non-zero, then either none or all of the valid History Buffer must be read.

4.4 Error Codes

4.4.1 Operation Status Codes

The operation status code for a descriptor is written to the Status field of the completion record for the descriptor if the Completion Record Address Valid flag in the descriptor is 1. If the operation status is 0x1a, 0x1b, or 0x1d, or if the Completion Record Address Valid Flag is 0 and the operation status is not equal to 0x01, then the operation status code is written to the SWERROR register instead.

The operation status codes are the same as for Intel DSA, with the exception of those listed in Table 4-4.

0x02	Unused.
0x05–0x09	Unused.
0x0a	Analytics error. A more specific code is in the Error Code field.
0x0b	Output buffer overflow. AECS is written if the Write Source 2 flag is non-zero.
0x11	Invalid flags. One or more flags in the descriptor Operations Flags field contain an unsupported or reserved value.
0x12	Non-zero reserved field (other than a flag in the Operations Flags field).

0x13	Invalid value for Source 1 Transfer Size, Source 2 Transfer Size, or Maximum Destination Size.
0x14-0x15	Unused.
0x17-0x18	Unused.
0x1b	Completion Record Address is not 64-byte aligned.
0x1c	Misaligned Address: The AECS address or size (as specified for Source 2) was not a multiple of 32 bytes.
0x23	Timeout waiting for response to a Page Request. The error is also recorded in SWERROR.
0x24	Watchdog timer expired without the device making progress.
0x30	Invalid Decompression/Compression/CRC Flags. A field in Decompression/Compression/CRC Flags contains an unsupported or reserved value.
0x31	Invalid Filter Flags. A field in Filter Flags contains an unsupported or reserved value.
0x32	Invalid Input Size. The input size for a Zcompress16, Zcompress32, Zdecompress16, or Zdecompress32 operation is not a multiple of the appropriate base size.
0x33	Invalid Number of Elements: Number of Elements is 0 for a filter operation.
0x34	Invalid Source-1 Width: For a Set-Membership or Find-Unique operation, the Source-1 Bit Width minus the Drop High Bits and Drop Low Bits was too large or too small. For RLE-Burst, the Source-1 Bit Width is not (logically) 8, 16, or 32.
0x35	Invalid Invert Output: The Invert Output flag was used when the output was not a bit-vector.

Table 4-4: Operation status codes

4.4.2 Error Code

When the Operation Status Code has the value 0x0a, the Error Code (byte 1 of the Completion Record) contains an error code that provides more detail on the type of error. The error codes are listed in Table 4-5.

Error Code	Detected Error	Description
0x01	Header too large to save/restore	Reached the end of the input stream before decoding header and header is too large to fit in input buffer.
0x02	Undefined CL code	Bad Code Length code, CL CAM is not hit, or code length of 0.
0x03	First code in LL tree is 16	
0x04	First code in D tree is 16	
0x05	No valid LL code	All of the codes are specified with 0 length.
0x06	Wrong number of LL codes	After parsing LL code lengths, total codes != expected value. Last CL code gave a repeat count that pushed the total above the expected value.
0x07	Wrong number of DIST codes	After parsing DIST code lengths, total codes != expected value. Last CL code gave a repeat count that pushed the total above the expected value.
0x08	Bad CL code lengths	First code of length N is greater than 2^N-1 or last code is greater than 2^7.
0x09	Bad LL code lengths	First code of length N is greater than 2^N-1 or last code is greater than 2^{15}.

Error Code	Detected Error	Description
0x0A	Bad DIST code lengths	First code of length N is greater than 2^N-1 or last code is greater than 2^{15}.
0x0B	Bad LL Codes	Bad Literal/Length Code neither ALUs nor EB CAM have hit, or 0 code length.
0x0C	Bad D Code	Bad Distance Code D CAM not hit, or 0 code length.
0x0D	Invalid Block Type	Block Type 0x3 detected.
0x0E	Invalid Stored Length	Length of stored block doesn't match inverse length.
0x0F	Bad End of File	End of file flag was set but last token was not EOB.
0x10	Bad Length Decode	Decoded Length is 0 or greater than 258.
0x11	Bad Distance Decode	Decoded Distance is 0 or greater than History Buffer Size.
0x12	Distance before Start of File	Distance of reference is before start of file.
0x13	Timeout	Engine has input data and room in the output buffer but is not making forward progress.
0x14	PRLE Format Error	PRLE record contains an error or is truncated.
0x15	Filter Function Word Overflow	Filter Function processing resulted in an output element that was too wide to fit into the specified output bit-width.
0x16	AECS Error	AECS contains an invalid value.
0x17	Source 1 Too Small	Source 1 contained fewer than expected elements.
0x18	Source 2 Too Small	Source 2 contained fewer than expected elements.
0x19	Unrecoverable Output Overflow	Output buffer was too small for generated output and the operation was not Decompress.
0x1A	Distance Spans Mini-blocks	During index generation as part of decompress, a match referenced data in a different mini-block.
0x1B	Length Spans Mini-blocks	During index generation as part of decompress, a match had a length extending into the next mini-block.
0x1C	Invalid Block Size	During index generation as part of decompress, a block header occurred that was not on a multiple of the mini-block size.
0x1D	Zcompress Verify Failure	The verify logic for Zcompress detected incorrect output.
0x1E	Invalid Huffman Code	A compression job tried to use a Huffman code with zero size.
0x1F	PRLE bit-width too large	The bit-width specified in the first byte of a PRLE stream was greater than 32.
0x20	Too Few Elements Processed	The input stream ended before specified Number of Input Element was seen.
0x21	Invalid RLE Count	For an RLE Burst operation with cumulative counts, the counts were seen to decrease.
0x22	Invalid Z-Decompress Header	During a Z-Decompress Operation, the input data ended in the middle of a record header.
0x23	Too Many LL Codes	The number of LL codes specified in the Deflate header exceeded 286.
0x24	Too Many D Codes	The number of D codes specified in the Deflate header exceeds 30.

Error Code	Detected Error	Description
0x25	Administrative Timeout	This operation was terminated because it exceeded a temporary timeout limit imposed by an administrative command. Resubmitting the descriptor is recommended.

Table 4-5: Error Codes

§

5 Software Architecture

5.1 Intel® Query Processing Library

The Intel Query Processing Library (QPL) provides user-mode access to the device, making its functions available to applications in a more user-friendly manner. The library provides functions for each operation type and provides both blocking and non-blocking modes of operation.

The library interfaces with the kernel-mode driver to request access to the hardware on behalf of the application. It normally services application requests using ENQCMD to a limited portal. If the ENQCMD fails due to congestion, the library may use a kernel-mode driver service to proxy the request to ensure forward progress. Additionally, the library can service application requests using MOVDIR64B to a dedicated work queue portal.

The library has two main purposes. One is to map from a user-friendly API to the device-centric data structures (e.g., Descriptors, Completion Records, etc.). The other is to provide necessary functionality that is not provided by the hardware. This includes such things as computing Huffman Tables and Deflate headers, and creating stored blocks when compressed data actually expands. It also orchestrates the flow to the hardware when multiple hardware invocations are needed for a single task. For example, dynamic compression with verify requires that the hardware be invoked three times: once to generate the statistics, once to do the compression, and once to perform the verification.

6 Structure Formats

6.1 Descriptor

An Intel IAA descriptor is a 64-byte structure that is submitted to a WQ portal to initiate an operation. The format of a descriptor is shown in Figure 6-1.

Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0	bytes
Operation	Operation Flags			Priv	Ignored	PASID		0
Completion Record Address								8
Source 1 Address								16
Destination Address								24
(De)compression Flags		Completion Interrupt Handle		Source 1 Transfer Size				32
Source 2 Address								40
Source 2 Transfer Size				Maximum Destination Size				48
Number of Input Elements				Filter Flags				56

Figure 6-1: Descriptor Format

The (De)compression Flags contain Decompression Flags if the Operation is Decompress or one of the Filter Operations (Scan ... Expand). It contains Compression Flags if the Operation is Compress. It is reserved for other operations.

6.1.1 Trusted Fields

Offset: 0; Size: 4 bytes (32 bits)

When a descriptor is submitted to an SWQ, these fields carry the Privilege and PASID of the software entity that submitted the descriptor. When a descriptor is submitted to a DWQ, these fields in the descriptor are ignored; the device uses the WQ Priv and WQ PASID fields of the WQCFG register.

On Intel CPUs, when software submits a descriptor to an SWQ using ENQCMD, these fields in the source descriptor are reserved. The value of IA32_PASID MSR is placed in the PASID field and the Priv field is set to 0 before the descriptor is sent to the device. When software uses ENQCMDS, these fields in the source descriptor must be initialized appropriately by software. If the Privileged Mode Enable field of the PCI Express PASID capability is 0, the Priv field must be 0.

Bits	Description
31	**Priv (User/Supervisor)** **0:** The descriptor is a user-mode descriptor submitted directly by a user-mode client or submitted by the kernel on behalf of a user-mode client. **1:** The descriptor is a kernel-mode descriptor submitted by kernel-mode software.
30:20	**Reserved**
19:0	**PASID** This field contains the Process Address Space ID of the requesting process.

Table 6-1: Descriptor Trusted Fields

6.1.2 Operation

Offset: 7; Size: 1 byte (8 bits)

This field specifies the operation to be executed.

0x00	No-op
0x01	Unused
0x02	Drain
0x03	Memory Move
0x04–0x3f	Unused
0x40	Reserved
0x41	Reserved
0x42	Decompress
0x43	Compress
0x44	CRC-64
0x45-0x47	Reserved
0x48	Zdecompress32
0x49	Zdecompress16
0x4a-0x4b	Reserved
0x4c	Zcompress32
0x4d	Zcompress16
0x4e–0x4f	Reserved
0x50	Scan
0x51	Set Membership
0x52	Extract
0x53	Select
0x54	RLE Burst
0x55	Find Unique
0x56	Expand

Table 6-2: Operation types

6.1.3 Operation Flags

Offset: 4; Size: 3 bytes (24 bits)

Operation Flags	
Bits	Description
23	**Reserved:** Must be 0.
22	**AECS R/W Toggle Selector** Let A be the Source 2 Address and S be the Source 2 Transfer Size. Then if source 2 is being read as AECS or being written: 0: Reads are done from (A) and writes are done to (A+S). 1: Reads are done from (A+S) and writes are done to (A). If Read Source 2 ≠ 1 and Write Source 2 = 0, then this field is reserved.
21	**CRC Select** **0:** Use the CRC polynomial 0x104c11db7, following ITU-T Recommendation V.42. **1:** Use the CRC polynomial 0x11edc6f41, following RFC 3720. This field is reserved for No-op, Drain, Memory Move, and CRC-64.

Operation Flags	
Bits	Description
20	Source 2 Steering Tag Selector Selects a steering tag entry from the TPH ST Table in the TPH Requester Capability to be used for writes to Source 2 Address. The meaning of the steering tags is platform dependent, but is expected to be programmed as follows: 0: Writes to the destination are not identified as writes to durable memory. 1: Writes to the destination are identified on the fabric as writes to durable memory. This field is reserved if the ST Mode Select field in the TPH Requester Control Register is 0. This field is reserved when Write Source 2 is 0.
19:18	Write Source 2 0: Source 2 is not written. 1: AECS written at completion of operation. 2: AECS written only if output overflow occurs. 3: Reserved. This field is reserved for operation types other than Decompress and Compress. A value of 2 is only allowed for Decompress. This field is reserved if Read Source 2 has a value of 2. This field is reserved if the operation is Compress and bit-0 of the Compression Flags (stats-mode) is set.
17:16	Read Source 2 0: Source 2 is not read. 1: Source 2 is read as AECS. 2: Source 2 is read as secondary input to filter function. 3: Reserved. This field is reserved for No-op, Drain, Memory Move, Zcompress32, Zcompress16, Zdecompress32, Zdecompress16, and CRC-64. The value 2 is required for Set Membership, Select, RLE Burst, and Expand. The value 2 is reserved for all other operation types. This field is reserved if the operation is Compress and bit-0 of the Compression Flags (stats-mode) is 1. The value 1 is required for Compress if bit-0 of the Compression Flags (stats-mode) is 0.
15	Destination Steering Tag Selector Selects a steering tag entry from the TPH ST Table in the TPH Requester Capability to be used for writes to Destination Address. The meaning of the steering tags is platform dependent, but is expected to be programmed as follows: 0: Writes to the destination are not identified as writes to durable memory. 1: Writes to the destination are identified on the fabric as writes to durable memory. This field is reserved if the ST Mode Select field in the TPH Requester Control Register is 0. This field is reserved for operation types that do not write to memory: No-op, Drain, and CRC-64.
14	Destination Readback 0: No readback is performed. 1: After all writes to the destination have been issued by the device, a read of the final destination address is performed before the operation is completed. The readback is performed only if the descriptor is completed successfully. This field is reserved if the Destination Readback Support field in GENCAP is 0. This field is reserved for No-op and Drain.

Operation Flags	
Bits	Description
13	Strict Ordering 0: Default behavior: writes to the destination can become globally observable out of order. The completion record write has strict ordering, so it always completes after all writes to the destination are globally observable. 1: Forces strict ordering of all memory writes produced by the device and ensures that they become globally observable in that order. This field is reserved for operation types that do not write to memory: No-op, Drain, and CRC-64. Note that this flag has nothing to do with the order in which descriptors are executed. It only affects ordering of the writes generated by this descriptor.
12	Completion Record TC Selector This field selects the Traffic Class value used for writing the completion record. It selects one of the two TC values in the Group Configuration Register corresponding to the WQ that the descriptor was submitted to. See the Intel DSA architecture specification for information on the use of Traffic Classes. 0: Use TC-A in the Group Configuration Register. 1: Use TC-B in the Group Configuration Register. This field is reserved when Completion Record Address Valid is 0.
11	Source 2 TC Selector This field selects the TC value used for reads and writes to Source 2 Address. It selects one of the two TC values in the Group Configuration Register corresponding to the WQ that the descriptor was submitted to. 0: Use TC-A in the Group Configuration Register. 1: Use TC-B in the Group Configuration Register. This field is reserved when Read Source 2 and Write Source 2 are both 0 and for operation types that do not use Source 2: No-op, Drain, Memory Move, Zcompress32, Zcompress16, Zdecompress32, Zdecompress16, and CRC-64.
10	Destination TC Selector This field selects the TC value used for writes to Destination Address. It selects one of the two TC values in the Group Configuration Register corresponding to the WQ that the descriptor was submitted to. 0: Use TC-A in the Group Configuration Register. 1: Use TC-B in the Group Configuration Register. For most operation types, this field selects the TC value used for writes to Destination Address. For Drain, this field selects the TC value used for readback from Readback Address 2 and is referred to as the Address 2 TC selector. This field is reserved for operation types that do not use Destination Address: No-op, and CRC-64.
9	Source 1 TC Selector This field selects the TC value used for reads from Source 1 Address. It selects one of the two TC values in the Group Configuration Register corresponding to the WQ that the descriptor was submitted to. 0: Use TC-A in the Group Configuration Register. 1: Use TC-B in the Group Configuration Register. For most operation types, this field selects the TC value used for reads from Source Address. For Drain, this field selects the TC value used for readback from Readback Address 1 and is referred to as the Address 1 TC selector. This field is reserved for operation types that do not use Source 1: No-op and Drain.

Operation Flags	
Bits	Description
8	**Cache Control** **0:** Hint to direct data writes to memory. **1:** Hint to direct data writes to CPU cache. This hint does not affect writing to the completion record, which is always directed to cache. If the Cache Control Support (Memory) field in GENCAP is 0, this field is reserved. This field is reserved for No-op, Drain, and CRC64.
7:6	**Reserved.** Must be 0.
5	**Completion Record Steering Tag Selector** Selects a steering tag entry from the TPH ST Table in the TPH Requester Capability to be used for writing the completion record. The meaning of the steering tags is platform dependent, but is expected to be programmed as follows: **0:** Writes to the destination are not identified as writes to durable memory. **1:** Writes to the destination are identified on the fabric as writes to durable memory. This field is reserved if the ST Mode Select field in the TPH Requester Control Register is 0 or if Completion Record Address Valid is 0.
4	**Request Completion Interrupt** **0:** No interrupt is generated when the operation completes. **1:** An interrupt is generated when the operation completes. If both a completion record and a completion interrupt are generated, the interrupt is always generated after the completion record is written. See the Intel DSA architecture specification for information regarding the interrupt to be generated. This field is reserved if User-mode Interrupts Enable is 0 and Priv is 0 (indicating a user-mode descriptor). If WQ PASID Enable control is 0, this field is not-reserved, independent of the setting of the User-mode Interrupts Enable control.
3	**Request Completion Record** **0:** A completion record is written only if the operation status is not equal to 0x01. **1:** A completion record is always written at the completion of the operation. This flag must be 1 for any operation other than No-op, Drain, and Memory Move. This flag must be 0 if Completion Record Address Valid is 0.
2	**Completion Record Address Valid** **0:** The completion record address is not valid. **1:** The completion record address is valid. This flag must be 1 for any operation other than No-op, Drain, and Memory Move.
1	**Block On Fault** **0:** Page faults cause partial completion of the descriptor. **1:** The device waits for page faults to be resolved and then continues the operation. This flag does not affect the handling of page faults on Completion Record Address or Drain Readback Address, all of which always block on fault. This field is reserved if the Block on Fault Enable field in WQCFG is 0. This field is reserved for certain operation types: No-op and Drain.
0	**Reserved.** Must be 0.

Table 6-3: Descriptor Flags

6.1.4 Decompression Flags

Offset: 38; Size: 2 bytes (16 bits)

All of these fields are reserved for the following operations: No-op, Drain, Memory Move, Zcompress32, Zcompress16, Zdecompress32, and Zdecompress16.

For the operation Compression, this contains the fields described in Section 6.1.5.

Decompression Flags	
Bits	Description
15:13	**Reserved.** Must be 0.
12:10	**Enable Indexing** **0:** Indexing is not enabled. **1:** Enable indexing with a mini-block size of 512 bytes. **2:** Enable indexing with a mini-block size of 1kB. **3:** Enable indexing with a mini-block size of 2kB. **4:** Enable indexing with a mini-block size of 4kB. **5:** Enable indexing with a mini-block size of 8kB. **6:** Enable indexing with a mini-block size of 16kB. **7:** Enable indexing with a mini-block size of 32kB. If this field is not 0, then Suppress Output must be set. This field is reserved for operations other than Decompress.
9	**Suppress Output** **0:** Decompressed / filter data is written to the output stream. **1**: Decompressed / filter data is not written to the output stream. This field is reserved for operations other than Decompress, Scan, Set Membership, Extract, Select, RLE Burst, Find Unique, and Expand.
8:6	**Ignore End Bits** Specifies the number of bits to ignore at the end of the compressed input stream. A value of 0 means that the entire last byte is processed, while a value of 7 means only one bit of the last input byte is processed.
5	**Decompress Bit Order** Specifies the bit order of the decompression input. **0:** Bit 0 of each 16-bit word is the least significant bit. (Little-endian. This is the normal format for a Deflate stream.) **1:** Bit 0 of each 16-bit word is the most significant bit and bit 15 is the least significant bit. (Big endian.)
4	**Select BFinal EOB** **0:** Any EOB block is treated as an appropriate EOB. **1:** Only EOB blocks with BFinal in the header are treated as an appropriate EOB.
3	**Check for EOB** **0:** Do not check whether the last token is an appropriate EOB. **1:** If the last token processed is not an appropriate EOB, Status is set to Analytics error and Error Code is set to Bad End of File.
2	**Stop on EOB** **0:** Do not stop processing when an appropriate EOB is detected. **1:** Stop processing when an appropriate EOB is detected.

Decompression Flags	
Bits	Description
1	**Flush Output** **0:** A partial output word is saved in the AECS. This value should be used for a Decompress descriptor that is part of a multiple-descriptor job and is not the last descriptor in the job. **1:** A partial output word is written to the output stream. If it would overflow the output buffer, it is saved in the AECS, so that the job can be completed by a subsequent descriptor. This value should be used for a Decompress descriptor that is the last (or only) descriptor in a job. For filter operations, output flushing is automatic and this flag is ignored. This flag must be 0 if Enable Indexing is non-zero and Write Source 2 is non-zero.
0	**Enable Decompression** **0:** Pass Source 1 directly to the filter function. **1:** Decompress Source 1 and pass the decompressed output to the filter function. If this field is 0, all other decompression flags except for Flush Output and Suppress Output are reserved. If Operation is Decompress, this field must be 1.

Table 6-4: Decompression Flags

6.1.5 Compression Flags

Offset: 38; Size: 2 bytes (16 bits)

For operations other than compress, these bits are interpreted according to Section 6.1.4.

Compression Flags	
Bits	Description
15:9	**Reserved.** Must be 0.
8:6	**Enable Indexing** **0:** No Indexing. **1:** Index every 512 bytes. **2**: Index every 1KB. **3**: Index every 2KB. **4**: Index every 4KB. **5**: Index every 8KB. **6**: Index every 16KB. **7**: Index every 32KB. When indexing is enabled, the input buffer size must be a multiple of the indexing size, unless Flush Output is 1 or Stats Mode is 1.
5	**Compress Bit Order** Specifies the bit order of the decompression input. **0:** Bit 0 of each 16-bit word is the least significant bit. (Little-endian. This is the normal format for a Deflate stream.) **1:** Bit 0 of each 16-bit word is the most significant bit and bit 15 is the least significant bit. (Big-endian.)
4	**Generate All Literals** **0:** Generate literals and matches. **1:** Generate only literals. This results in only doing Huffman Compression.

Compression Flags	
Bits	Description
3:2	End Processing 0: Append nothing after final output token. 1: Append EOB after final token. 2: Append EOB and non-bFinal Stored Block after final token. 3: Append EOB and bFinal Stored Block after final token.
1	Flush Output 0: A partial output word is saved in the AECS. This value should be used for a Compress descriptor that is part of a multiple-descriptor job and is not the last descriptor in the job. 1: A partial output word is written to the output stream. This value should be used for a Compress descriptor that is the last (or only) descriptor in a job.
0	Stats Mode 0: Generate Huffman output. 1: Generate Statistics output.

Table 6-5: Compression Flags

6.1.6 Filter Flags

Offset: 56; Size: 4 bytes (32 bits)

All of these fields are reserved for the following operations: No-op, Drain, Memory Move, Decompress, Compress, Zcompress32, Zcompress16, Zdecompress32, and Zdecompress16.

Filter Flags	
Bits	Description
31:27	Reserved: Must be 0.
26:22	Drop High Bits For the Set Membership and Find Unique operations, this field specifies the number of most significant bits of the input to discard. This field is reserved for other operation types.
21:17	Drop Low Bits For the Set Membership and Find Unique operations, this field specifies the number of least significant bits of the input to discard. This field is reserved for other operation types.
16	Invert Output 0: The bits of the output are not inverted. 1: For operations whose output is a bit vector, each bit of the output is inverted. This field is reserved for operation types whose output is an array with width greater than 1.
15	Output Bit Order 0: Bit 0 of each output byte is the LSB. (Little-endian.) 1: Bit 7 of each output byte is the LSB. (Big-endian.)

Filter Flags	
Bits	Description
14:13	Output Width 0: The output of the filter is unmodified; depending on the operation, the output is either a bit vector or an array whose elements have the same width as the input. 1: The output elements are 8 bits. 2: Output elements are 16 bits. 3: Output elements are 32 bits. If this field is non-zero and the default filter output is an array, each element of the array is zero-extended to the specified width. The specified width must be greater than or equal to the width of the primary input. If this field is non-zero and the default filter output is a bit vector, the output is modified to an array of indices of the 1 bits in the bit vector. Each index has the specified width, which must be sufficient to represent the maximum index value.
12	Source 2 Bit Order 0: Bit 0 of each Source 2 byte is the LSB. (Little-endian.) 1: Bit 7 of each Source 2 byte is the LSB. (Big-endian.) This field is used only when Read Source 2 is 2; otherwise, it is reserved.
11:7	Source 2 Width This field indicates the size in bits of the data elements in the secondary input stream. The element width is the value in this field plus 1. This field is used for RLE Burst. It is reserved for all other operation types, where the secondary input stream is a bit vector.
6:2	Source 1 Width This field indicates the size in bits of the data elements in the primary input stream. The element width is the value in this field plus 1. The specified element width must be greater than the sum of Drop Low Bits and Drop High Bits. If Source 1 Parser is Parquet RLE, this field is reserved, because the field width is specified in the header of the input stream.
1:0	Source 1 Parser 0: The input consists of a packed array of values in little-endian format with the bit width given by Source 1 Width. 1: The input consists of a packed array of values in big-endian format with the bit width given by Source 1 Width. 2: The input is in the Parquet RLE format, as described in 3.6.3. 3: Reserved.

Table 6-6: Filter Flags

6.1.7 Completion Record Address

Offset 8; Size 8 bytes (64 bits)

This field specifies the address of the completion record. The completion record is 64 bytes and must be aligned on a 64-byte boundary. If the Completion Record Address Valid flag is 0, this field is reserved.

If the Request Completion Record flag is 1, a completion record is written to this address at the completion of the operation. If Request Completion Record flag is 0, a completion record is written only if there is an error.

The Completion Record Address Valid and Request Completion Record flags must both be 1 and the Completion Record Address must be valid for any operation other than No-op, Drain, and Memory Move.

6.1.8 Source 1 Address

Offset: 16; Size: 8 bytes (64 bits)

This field specifies the address of the primary source data. This field is reserved for No-op. The value of this field is ignored if Source 1 Transfer Size is zero. If the Source Address and Transfer Size are not both aligned to a multiple of 64 bytes, an implementation may read more source data than required by the descriptor. For example, source data may be read in aligned 32-byte chunks. The excess data is discarded.

6.1.9 Destination Address

Offset: 24; Size: 8 bytes (64 bits)

This field specifies the address of the destination buffer.

The destination buffer must not overlap the source 1 buffer. It must not overlap the source 2 buffer if either the Read Source 2 or Write Source 2 flag is non-zero. For the purpose of this check, the destination buffer size is Source 1 Transfer Size if the operation type is Memory Move; otherwise, the destination buffer size is Maximum Destination Size.

This field is reserved for No-op, and CRC-64.

This field is ignored if Maximum Destination Size is zero.

6.1.10 Source 1 Transfer Size

Offset: 32; Size: 4 bytes (32 bits)

This field indicates the number of bytes to be read from the source 1 address to perform the operation. This field is reserved for No-op and Drain.

The maximum allowed transfer size is dependent on the WQ the descriptor was submitted to. It is specified by the WQ Maximum Transfer Size field for the WQ in the WQ Configuration Table (which is, in turn, limited by the Maximum Supported Transfer Size field in the General Capabilities Register).

For the operations Zcompress16 and Zdecompress16, the size must be a multiple of 2.

For the operations Zcompress32 and Zdecompress32, the size must be a multiple of 4.

For the operation Compress, if Enable Indexing is not 0, the size must be a multiple of specified mini-block size, unless Flush Output or Stats Mode is set.

If the operation is Memory Move, this field must be non-zero. Otherwise, if the operation is other than No-op or Drain, then one of Source 1 Transfer Size and Maximum Destination Size must be non-zero.

6.1.11 Completion Interrupt Handle

Offset: 36; Size: 2 bytes (16 bits)

This field specifies the interrupt table entry to be used to generate a completion interrupt. See the Intel DSA architecture specification for details.

This field is reserved if the Request Completion Interrupt flag is 0.

6.1.12 Source 2 Address

Offset: 40; Size: 8 bytes (64 bits)

This field specifies the address of either the AECS or the secondary input to the filter function, depending on the values of the Read Source 2 and Write Source 2 flags. If this field specifies the address of the AECS, it may be read or written or both (despite the field name). If this field specifies the address of the AECS, then its value (the address) must be 32-byte aligned.

If the Write Source 2 flag is non-zero, the source 2 buffer must not overlap the source 1 buffer.

This field is reserved if the operation type is No-op, Drain, Memory Move, CRC-64, Zcompress32, Zcompress16, Zdecompress32, or Zdecompress16, or if Read Source 2 and Write Source 2 are both 0.

This value is ignored if Source 2 Transfer Size is zero.

6.1.13 Maximum Destination Size

Offset: 48; Size: 4 bytes (32 bits)

This field indicates the maximum size of the output buffer. The maximum allowed size is specified by the WQ Maximum Transfer Size field for the WQ in the WQ Configuration Table.

This field is reserved if the operation type is No-op, Drain, Memory Move, or CRC-64.

If the operation is other than No-op or Drain, then one of Source 1 Transfer Size and Maximum Destination Size must be non-zero.

6.1.14 Source 2 Transfer Size

Offset: 52; Size: 4 bytes (32 bits)

This field indicates the size of the source 2 buffer. The maximum allowed size is specified by the WQ Maximum Transfer Size field for the WQ in the WQ Configuration Table.

If Source 2 is an AECS, then the transfer size must be a non-zero multiple of 32-bytes. If Source2 is Filter Input Data, then the transfer size must be non-zero.

This field is reserved if the operation type is No-op, Drain, Memory Move, CRC-64, Zcompress32, Zcompress16, Zdecompress32, or Zdecompress16 or if Read Source 2 and Write Source 2 are both 0.

If the operation is Compress and bit-0 of the Compression Flags (stats-mode) is 0, then the source-2 transfer size must be 1,568 bytes.

If the operation is Compress and Write Source 2 is not 0, then the amount of data written is 64 bytes.

6.1.15 Number of Input Elements

Offset: 60; Size: 4 bytes (32 bits)

This field is used to determine the end of the input stream for filter operations. Since the input elements are packed and may be smaller than 1 byte, the number of elements cannot always be determined from the number of bytes of input. This field indicates the number of elements in the primary input stream (after decompression, if applicable), except for the Expand operation, where it specifies the number of bits in the secondary input stream.

For the operations where this field is not reserved, it must have a non-zero value.

For the operation RLE-BURST, if the source1-width is logically 32 (i.e., the source1-width field in the Filter Flags is 31), then the Number of Input Elements must be at least 2.

This field is reserved for the following operation types: No-op, Drain, Memory Move, Decompress, Compress, Zcompress32, Zcompress16, Zdecompress32, or Zdecompress16.

6.2 Completion Record

The completion record is a 64-byte structure in memory that the device writes when an operation is complete or encounters an error. A completion record address is in each descriptor. The completion record address must be 64-byte aligned. Software should not depend on the value of unused fields (including fields that are unused for specific operation types).

<table>
<tr><th>Byte 7</th><th>Byte 6</th><th>Byte 5</th><th>Byte 4</th><th>Byte 3</th><th>Byte 2</th><th>Byte 1</th><th>Byte 0</th><th>bytes</th></tr>
<tr><td colspan="4">Bytes Completed</td><td colspan="2">Unused</td><td>Error code</td><td>Status</td><td>0</td></tr>
<tr><td colspan="8">Fault Address</td><td>8</td></tr>
<tr><td colspan="4">Unused</td><td colspan="4">Invalid Flags</td><td>16</td></tr>
<tr><td colspan="2">XOR Checksum</td><td>Unused</td><td>Output Bits</td><td colspan="4">Output Size</td><td>24</td></tr>
<tr><td colspan="4">Min / First</td><td colspan="4">CRC</td><td>32</td></tr>
<tr><td colspan="4">Sum / Population count</td><td colspan="4">Max / Last</td><td>40</td></tr>
<tr><td colspan="8" rowspan="2">Unused</td><td>48</td></tr>
<tr><td>56</td></tr>
</table>

Figure 6-2: Completion Record Format

6.2.1 Status

Offset: 0; Size: 1 byte (8 bits)

This field reports the completion status of the descriptor. Hardware never writes 0 to this field. Software should initialize this field to 0 so it can detect when the completion record has been written. See section 4.4.1 for a list of the operation status codes and their meanings.

Bits	Description
7	**R/W** (Not used unless Operation Status indicates a translation fault – code 0x03, 0x04, or 0x1f). **0:** The faulting access was a read. **1:** The faulting access was a write.
6	Unused.
5:0	**Operation Status** See section 4.4.1 for the meaning of the value in this field.

Table 6-7: Completion record Status field

6.2.2 Error Code

Offset: 1; Size: 1 byte (8 bits)

When the Status field is equal to 0x0Ala, this field indicates the type of error. If Status has any other value, this field is unused. Software should not depend on the value of this field for operation types where it is unused. See section 4.4.2 for a listing of the error codes.

6.2.3 Bytes Completed

Offset: 4; Size: 4 bytes (32 bits)

This field can be used in some cases to continue a partially executed operation. In particular:

- If an operation terminated due to a page fault, this field contains the number of bytes successfully written to the output.

- If a Decompress operation terminates with an Output Buffer Overflow status, this field contains the number of bytes that were consumed from Source-1.
- If the operation fully completed with success, this field contains 0.
- Otherwise (e.g., if the operation terminated due to some other error), this field is undefined.

The only partially processed operations that can be successfully continued from where it left off are:

- A Decompress operation that resulted in an Output Buffer Overflow.
- A Memory Move operation that resulted in a page fault[1].

A page fault is indicated by Operation Status codes 0x03, 0x04, or 0x1f.

6.2.4 Fault Address

Offset: 8; Size: 8 bytes (64 bits)

If the operation terminated due to a page fault, this field contains the address that caused the fault. Bits 11:0 may be reported as 0. Page faults are indicated by Operation Status codes 0x03, 0x04, and 0x1f, described in Table 4-4. For other errors, this field is undefined

6.2.5 Invalid Flags

Offset: 16; Size: 4 bytes (32 bits)

If the Operation Status is Invalid Operation Flags, Invalid Decompression Flags, Invalid Compression Flags, Invalid CRC Flags, or Invalid Filter Flags, this field contains a bitmask of the flags field that was found to be invalid, to aid in debugging. If a bit in this field is 1, it indicates that the flag at the corresponding bit position in the flags field of the descriptor was invalid. The implementation is not obligated to indicate every invalid flag that may be present in the descriptor, but it must indicate at least one anytime it reports an invalid flags error code.

If the operation status is anything other than Invalid Operation Flags, Invalid Decompression Flags, Invalid Compression Flags, Invalid CRC Flags, or Invalid Filter Flags, this field is unused.

6.2.6 Output Size

Offset: 24; Size: 4 bytes (32 bits)

This field contains the number of bytes written to the destination buffer. This field is not used for the following operation types: No-op, Drain, or Memory Move.

6.2.7 Output Bits

Offset: 28; Size: 1 byte (8 bits)

This field contains the number of bits written to the last byte of the destination. If this field is 0, all bits were written. This value should be used to determine the number of output elements generated when the output width is less than 8. This field is not used for the following operation types: No-op, Drain, or Memory Move.

[1] In the current implementation, operations other than Memory Move that result in a page fault set the Bytes Completed field to 0. These operations must be restarted from the beginning. This may change in future implementations.

6.2.8 XOR Checksum

Offset: 30; Size: 2 bytes (16 bits)

This field contains the XOR checksum computed on the uncompressed data (either the output of the decompressor, or the primary input when the Enable Decompression flag is 0). For the purpose of computing this checksum, the data is treated as 16-bit words. If there are an odd number of bytes, the final byte is zero-extended to 16 bits. This field is not used for the following operation types: No-op, Drain, Memory Move, CRC-64.

6.2.9 CRC

Offset: 32; Size: 4 bytes (32 bits)

This field contains the CRC computed on the uncompressed data (either the output of the decompressor, or the primary input when the Enable Decompression flag is 0). This field is not used for the following operation types: No-op, Drain, Memory Move, CRC-64.

6.2.10 Aggregates

Offset: 36; Size: 12 bytes (3 × 32 bits)

These fields contain information about the output, as follows:

Field	Byte Offset	Value when output is an array	Value when output is a bit vector
Min / First	32	Minimum value in output.	Index of first 1 bit in output.
Max / Last	36	Maximum value in output.	Index of last 1 bit in output.
Sum / Population count	40	Sum of all output values.	Number of 1 bits in output.

Table 6-8: Completion record Aggregates fields

These fields are not used for the following operation types: No-op, Drain, Memory Move, Compress, Decompress, Zcompress16, Zcompress32, Zdecompress16, Zdecompress32.

6.3 CRC-64 Descriptor and Completion Record

The CRC-64 operation, 0x44, uses a modified descriptor. The format is shown in Figure 6-3.

Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0	bytes
Operation	Operation Flags			Priv	Reserved	PASID		0
Completion Record Address								8
Source 1 Address								16
Reserved								24
CRC Flags	Completion Interrupt Handle			Source 1 Transfer Size				32
Reserved								40
								48
CRC Polynomial								56

Figure 6-3: CRC-64 Descriptor Format

All except for the CRC Flags and CRC Polynomial are the same as for the normal descriptor. Usage of these fields is described in Section 3.8.3.

The CRC Flags and CRC Polynomial are described in Sections 6.3.1 and 6.3.2.

The CRC-64 operation uses a modified Completion record. The format is shown in Figure 6-4.

Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0	bytes
Bytes Completed				Unused		Error code	Status	0
Fault Address								8
Unused				Invalid Flags				16
								24
								32
CRC-64								40
Unused								48
								56

Figure 6-4 : CRC-64 Completion Record Format

All of the fields other than CRC-64 is the same as in the normal Completion Record.

The CRC-64 field contains the CRC result, in the most-significant part of the field, as defined by the CRC Bit Order flag. See Section 3.8.3 for details.

6.3.1 CRC Flags

Offset: 38; Size: 2 bytes (16 bits)

CRC Flags	
Bits	Description
15	**CRC Bit Order** Specifies the bit order of the CRC input. **0:** Bit 0 of each byte is the least significant bit. **1:** Bit 0 of each byte is the most significant bit.
14	**Invert CRC** If this value is 0, then the initial value of the CRC is 0, and the residue is returned. If this value is 1, then the initial value of the CRC is all 1's, and the inverse of the residue is returned.

CRC Flags	
Bits	Description
13:0	**Reserved.** Must be 0.

6.3.2 CRC Polynomial

Offset: 56; Size: 8 bytes (64 bits)

This field defines the polynomial for the CRC, as described in Section 3.8.3. The polynomial is described in its normal (not bit-reversed) form, without the leading 1-bit, in the high order end of this field, so that bit-63 is the most significant bit.

For example, a 64-bit CRC would use all of the bits. A 32-bit CRC would place the polynomial in bits 63:32 of the field, and bits 31:0 would be 0.

The size of the polynomial is defined by the least significant 1-bit in this field. If all of the bits are 0, then the results are undefined.

6.4 Analytics Engine Configuration and State

The AECS structure contains parameters and internal state of the analytics engine. See section 3.2 for more information. The AECS has one of two different formats: one for decompression, and filter opcodes, and one for the compression opcode.

6.4.1 AECS Format for Decompress and Filter

The format of the AECS for decompress and filter is shown in Figure 6-5.

Byte 3	Byte 2	Byte 1	Byte 0	Byte Offset	Category
CRC				0x0	Filter
Reserved		XOR Checksum		0x4	
Low Filter Parameter				0x8	
High Filter Parameter				0xC	
Output Modifier Index				0x10	
Reserved		Drop Initial Decompress Out Bytes		0x14	
Reserved				0x18	Reserved
				0xA4	
Output Accumulator Data				0xA8	Decompress
				0xAC	
Reserved			Output Bits Valid	0xB0	
Bit Offset for Indexing				0xB4	
input Accumulator Data				0xB8	
				0x1B4	
Size QW 3	Size QW 2	Size QW 1	Size QW 0	0x1B8	
Size QW 31	Size QW 30	Size QW 29	Size QW 28	0x1D4	
Decompression State				0x1D8	
				0x14E4	

Figure 6-5: AECS Format for Decompress and Filter

CRC	On input, this field contains the CRC seed. On output, it is the CRC value. Note that these values are inverted as is specified by the relevant CRC standards.
XOR Checksum	Initial (on input) or final (on output) XOR Checksum value.
Low Filter Parameter	Low Parameter value of filter functions.
High Filter Parameter	High Parameter value of filter functions.
Output Modifier Index	Base index associated with first output bit. When the output is a bit-vector that is being modified, this value offsets the indices written to the output and the values aggregated.
Drop Initial Decompress Out Bytes	The number of initial bytes in the decompressed output that should be dropped before starting the filter operation.
Output Accumulator Data	The output accumulator is 8 bytes in size.

Output Accumulator Bits Valid	Number of valid data bits in output accumulator max 63.
Bit Offset for Indexing	Total number of consumed bits on input.
Input Accumulator Data	The input accumulator is 256 bytes in size.
Size QWn	The number of bytes valid in the corresponding Quadword in the Input Accumulator. Valid values are 0–64.
Analytics Internal State	Contains internal state of the Analytics hardware required to link together multiple segments that belong to one logical file.

Table 6-9: AECS fields for Decompress and Filter

6.4.1.1 Decompress Internal State

The Decompress/Analytics Internal State is shown in Table 6-10.

Field	Byte Offset
EOB CAM Entry	0x1D8
Reserved	0x1DC
ALU First Table Index	0x1E0–0x1F0
ALU Num Codes	0x1F4–0x204
ALU First Code	0x208–0x218
ALU First Len Code	0x21C–0x22C
LL CAM Entries	0x230–0x280
Reserved	0x284
LL CAM Total Lengths	0x288–0x294
Distance CAM Entries	0x298–0x30C
Distance CAM Total Lengths	0x310–0x320
Min Length-Code Length	0x324
LL Mapping Table	0x328–0x430
Decompress State	0x434
Stored Block Bytes Remaining	0x438
Reserved	0x43C–0x4E0
History Buffer Write Pointer	0x4E4
History Buffer	0x4E8–0x14E4

Table 6-10 Decompress/Analytics Internal State

The format of the CAM entries is the following: Bits 14:0 give the code value, in non-bit-reversed form, in the high-order bits of the field. Bit 15 is a valid bit, which should be set only for valid entries. Bits 30:16 give a bit-mask, where the 1-bits correspond to the valid bits in the code. If the size of the code is N, then the high-order N bits of the bit-mask should be 1, and the remaining bits should be 0.

The 5 DWORDs for each ALU field define 15 values, one for each of 15 ALUs, where each ALU is looking for codes of a particular width. E.g., ALU-1 is looking for codes that are 1-bit wide, and ALU-5 is looking for codes that are 5-bits wide. The way that these 15 fields are spread throughout the 5 DWORDS is shown in Table 6-11. The value in each field is stored towards the least-significant end.

ALU	DWORD	Bits
1	0	1:0
2	0	4:2
3	0	8:5
4	0	13:9
5	0	19:14
6	0	26:20
7	1	7:0
8	1	16:8
9	1	26:17
10	2	10:0
11	2	22:11
12	3	12:0
13	3	26:13
14	4	14:0
15	4	30:15

Table 6-11 ALU Field Definitions

The basic idea behind each ALU is that (in the non-bit-reversed space) all of the codes of a given length sequential values. So, these can be represented as a first code and a number of codes. There is a mapping table that maps to actual literal values. All of the literal values in that mapping table that correspond to codes of a given length are found in sequential locations. The ALU needs to know where in the mapping table corresponds to the first code. For example, ALU-4 might contain 3 codes starting with 0x8, and with a first table index of 10. This means that the valid length-4 codes are 0x8, 0x9, and 0xA. And these correspond to the literals in the mapping table entries 10, 11, and 12.

The actual fields are:

EOB CAM Entry	Huffman code for EOB token.
ALU First Table Index	Index into mapping table for first code for this ALU.
ALU Num Codes	Number of codes for this ALU.
ALU First Code	First code value for this ALU.
ALU First Len Code	First LL code value that is a length.
LL CAM Entries	These 21 CAM entries contain the codes for LL tokens 265–285.
Reserved	
LL CAM Total Lengths	These contain the total lengths (code length plus number of extra bits) for LL tokens 265–285. There are six lengths per DWORD, in bits: 4:0, 9:5, 14:10, 20:16, 25:21, and 30:26. The last DWORD only contains three lengths.
Distance CAM Entries	This CAM contains the Huffman codes for the 30 Distance tokens.
Distance CAM Total Lengths	This contains the total lengths for the distance tokens in the same format as for the LL CAM Total Lengths.
Min Length-Code Length	This contains the total length (code length plus number of extra bits) of the shortest Length Code. If there are no length codes, the value is 15.

LL Mapping Table	Each byte entry contains the value as indexed by the ALUs. For literals, the value is the literal value. For lengths, the value is the length – 3.
History Buffer Write Pointer	Bits 14:0 contain the write pointer into the history buffer. Bit 15 is set when the write pointer becomes greater or equal to the size of the history buffer.
Stored Block Bytes Remaining	Bits 14:0 indicate the number of remaining bytes in a stored block that has been partially processed. A value of 0 means that the parsing is not in the middle of a stored block.
Decompress State	Bits 3:0 indicate the state of the decompress parser. The possible values are: 0000: Looking at an LL token in a non-final block. 0100: Look at an LL token in a final block. 0010: Looking at a stored block byte in a non-final block. 0110: Looking at a stored block byte in a final block. 0XX1: Looking at the start of a block header. 1XXX: Processing terminated due to EOB. Bits 31:16 give the number of bits since the last token processed to the end of the input accumulator.
History Buffer	History buffer.

6.4.2 AECS Format for Compress

The format of the AECS for compress is shown in Figure 6-6.

Byte 3	Byte 2	Byte 1	Byte 0	Byte Offset	Category
CRC				0x0	Checksums
Reserved		XOR Checksum		0x4	
Reserved				0x8	
Reserved				0x18	
Reserved		Num Acc Bits Valid		0x1C	Output Accumulator
Output Accumulator Data				0x20	
				0x11C	
Huffman Literal Code 0				0x120	Huffman Tables
Huffman Literal Code 285				0x594	
Reserved				0x598	
Reserved				0x59C	
Huffman Distance Code 0				0x5A0	
Huffman Distance Code 29				0x614	

Figure 6-6 AECS Format for Compress

CRC	On input, this field contains the CRC seed. On output, it is the CRC value.
XOR Checksum	Initial (on input) or final (on output) XOR Checksum value.
Output Accumulator Data	On input, this field can contain up to 2,048 bits. On output, it will contain fewer than 64. This can be used by software to insert a Deflate block header.

Num Acc Bits Valid	Number of bits that are valid in Output Accumulator Data.
Huffman Codes	Bits 14:0: Non-bit-reversed Huffman code stored in low-order bits of field. Bits 18:15: Length of code word in bits. Bits 31:19: Reserved.

Table 6-12 AECS Fields for Compress

§

7 Summary of Differences from Intel® DSA

7.1 General Differences

The following lists some of the key aspects of Intel IAA that are different from Intel DSA:

- Partial descriptor completion: In general, Intel IAA completes the entire operation or returns an error. In case of an error, the entire operation needs to be re-executed. The two exceptions to this are the decompress and memory move operations. In the case of decompression where the output buffer is not large enough, the operation would result in a recoverable "output overflow". In this case, the operation is partially completed, the accelerator finishes in a clean state, and the job can be continued (in a new descriptor) where it left off. In the case of memory move, if a page fault occurs and Block on Fault is not set, then the operation is partially completed and can be continued where it left off.
- Batch processing: Intel IAA does not support batch processing or the Fence operation flag.
- Stateless device: There is no state information stored within the accelerator between operations. Any state information that needs to be passed between operations is written to the AECS by the first operation and then read from the AECS by the second operation.
- Read Buffer Allocation: Intel IAA does not support Read Buffer allocation control.
- Completion Records: The Intel IAA Completion Record is 64 bytes in length and must be aligned on a 64-byte boundary.
- Overlapping Buffers: None of the Source 1, Source 2, and Destination buffers (which are present for a given operation) may overlap. In particular, Memory Move does not support overlapping buffers.
- Performance Monitoring Events:
  - Intel IAA does not support the Operations events or Batch-related events.
  - In Intel IAA, the EV_CL_WRITE event measures total data written, in units of 512 bytes rather than units of 32 bytes.
- Operations: The operations supported by Intel IAA are different from those supported by Intel DSA. See section 6.1.2 for a list of the operations supported. The OPCAP register allows runtime detection of supported operations.
- CRC Operation: The CRC64 operation in Intel IAA is different from the CRC operation in Intel DSA. In particular, CRC64 supports an arbitrary polynomial up to degree 64. In general, implementations for fixed CRCs may be faster than implementations for arbitrary polynomials.
- Completion Record: Byte 1 has a different meaning. In Intel DSA, it is an operation-specific result, for Intel IAA, it is either not used or is an error code.

7.2 Configuration and Control Register Differences

Note that bits that are the same between Intel IAA and Intel DSA are not listed here. That is, any bit not listed here is the same as for Intel DSA.

7.2.1 General Capabilities Register (GENCAP)

GENCAP Base: BAR0		Offset: 0x10	Size: 8 bytes (64 bits)
Bit	Attr	Size	Description
63:52	RO	12 bits	Unused.
51:47	RO	5 bits	Maximum Set Size The maximum logical source-1 bit-width for the set-membership and find-unique operations is the value of this field plus 1. The logical source-1 bit-width is the actual source-1 bit-width minus the values of the Drop Low Bits and Drop High Bits fields in the Filter Flags.
46:42	RO	5 bits	Maximum Decompress Set Size The maximum logical source-1 bit-width for the set-membership and find-unique operations when source-1 is being decompressed is the value of this field plus 1. The logical source-1 bit-width is the actual source-1 bit-width minus the values of the Drop Low Bits and Drop High Bits fields in the Filter Flags.
41	RO	1 bit	Indexing Support 0: Indexing (for compress / decompress) is not supported. 1: Indexing is supported.
40	RO	1 bit	Decompress Support 0: Decompression is not supported 1: Decompression is supported.

7.3 PCI Express Configuration Register Differences

7.3.1 Device ID (DID)

DEVICE ID (DID) Identifies the particular device. Base: Rootbus CFG Offset: 0x2 Size: 2 bytes (16 bits) Default Value: 0x0CFE				
Bits	**Attr**	**Size**	**Default Val**	**Description**
15:0	ROS	16	0x0CFE	**Device ID (DID)** Allocated by the vendor.

7.3.2 Outstanding Page Request Capacity (PRSREQCAP)

OUTSTANDING PAGE REQUEST CAPACITY (PRSREQCAP)
Maximum Number of Outstanding Page Requests
Base: Rootbus CFG Offset: 0x248 Size: 4 bytes (32 bits)
Default Value: 0x00000100

Bits	Attr	Size	Default Val	Description
31:0	RO	32	0x100	**Capacity (CAP)** How many Page Requests can the function issue.

§