INTEL® IN-MEMORY ANALYTICS ACCELERATOR (INTEL® IAA) ARCHITECTURE SPECIFICATION

Document ID: 60941
Revision: 03
August 2023

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 the 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.

Revision History

Date	Revision Description
April 2022	The initial release of the document.
December 2022	First revision and refinement of content.
April 2023	• Removed “Intel Confidential” from header. • Changed Metadata: added Intel Corp as author, added keywords and summary. • Removed duplicated section from Chapter 6.
August 2023	Added discussions of index table and decompression on bit-boundaries.

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 designed to accelerate streaming operations such as memory copy and others.
QPL	Intel® Query Processing Library	Intel library to interface between applications and the hardware.

Table of Contents

List of Tables

List of Figures

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

Table 1-1 References

Intel® Data Streaming Accelerator Architecture Specification
Intel® 64 and IA-32 Architectures Software Developer's Manuals
Intel® Architecture Instruction Set Extensions Programming Reference
Intel® Query Processing Library
PCI Express* Base Specification 4.0
Intel® Virtualization Technology for Directed I/O Specification
Intel® Scalable I/O Virtualization Technical Specification
Intel® I/O Acceleration Technology
ITU-T Series V: Data Communication Over the Telephone Network-- Error Control V.42
RFC 1951, Deflate Compressed Data Format Specification
RFC 3720, Internet Small Computer Systems Interface

2 OVERVIEW

The Intel® In-Memory Analytics Accelerator (Intel® IAA) is a hardware accelerator that provides very high throughput compression and decompression combined with analytics 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 analytics 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 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 logically contains three main functional blocks: Compression, Encryption, and Analytics. The Analytics pipe contains three sub-blocks: Decrypt, Decompress, and Filter. These functions are tied together so each analytics operation can perform any combination of decrypt/decompress/filter (e.g., decrypt-filter), as illustrated in Figure 2-1. Alternatively, one can compress or encrypt the input. Compression and Encryption cannot be linked with any other operations.

Not all of the functional blocks and features are available on all versions of the accelerator. The availability of these blocks and features are conveyed through the capability registers, in particular OPCAP, GENCAP (Section 7.2.1), and IAACAP (Section 7.2.2). Software should consult these registers before using a capability described by one of these registers.

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 the use of memory bandwidth for uncompressed raw data transfer.

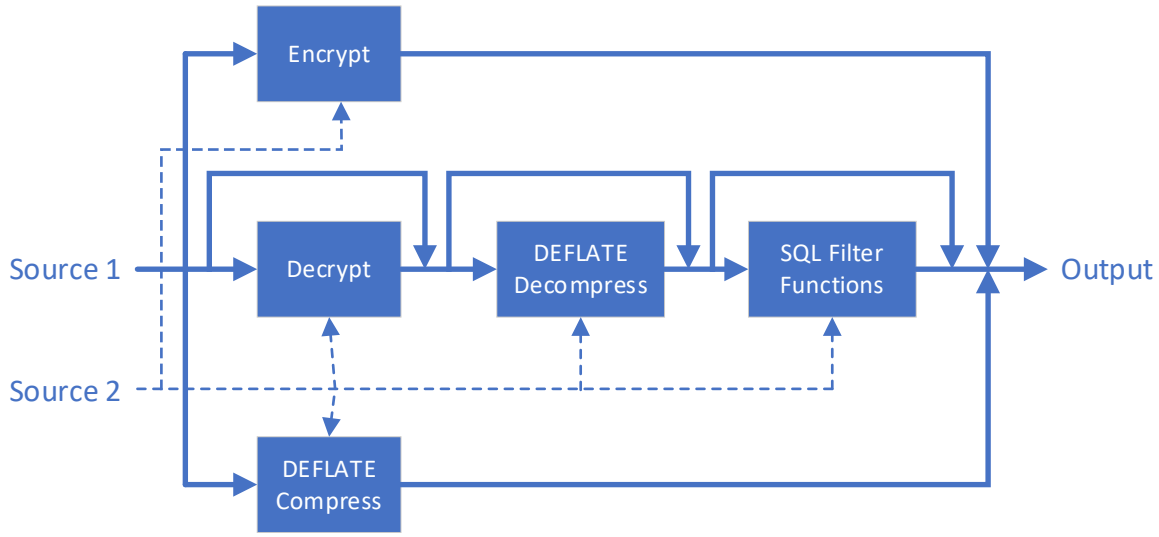

Figure 2-1 Intel® Analytics Accelerator

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 bytes.

It also supports Deflate compression, along with the calculation of arbitrary CRCs.

Intel IAA encryption/decryption supports the following algorithms: GCM-AES128 (GCM with 128-bit keys), GCM-AES256, CFB-AES128, CFB-AES256, XTS-AES128, and XTS-AES256.

The SQL filter function block takes one or two input streams, a primary input, and in some cases, a secondary input. The primary input may be read from memory or received from the decompression or decryption block. If used, the secondary input is always read from memory. The data streams logically contain an array of unsigned values; however, 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 and an XOR checksum of the data stream. See Section 3.6 for details. It also computes 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 accelerator supports data operations listed below, further detailed in the following sections.

Table 3-1 Supported Data Operations

Type	Operation	Description
Decompress	Decompress	Decompresses input data.
Compress	Compress	Compresses input data.
Decrypt	Decrypt	Decrypts input data.
Encrypt	Encrypt	Encrypts input data.
Filter	Scan	Computes a bit-mask of which entries satisfy a condition.
	Extract	Returns entries as specified by a range of entry indices.
	Select	Returns entries as specified by a bit-mask.
	Expand	Inserts zeros as specified by a bit-mask.
CRC	CRC64	Computes an arbitrary CRC up to 64-bits in size.
Memory	Translation Fetch	Prefetches address translations.

The analytics pipeline consists of three stages:

1. Decrypt
2. Decompress
3. Filter, CRC64

In general, any non-empty subset of these operations can be performed in this order. All other operations must be done individually.

The paradigm for configuring the decrypt/decompress/filter pipeline is that the opcode specifies the last operation to be performed, and earlier operations are enabled via flag bits. For example, if any filter operation is to be performed, the opcode specifies the filter operation and flag bits indicate whether decrypt and/or decompress is to be done. If decompression but no filter operation is to be performed, the decompress opcode is used. Only if decrypt is to be performed with no decompress and no filter operation is the decrypt opcode used.

3.2 Analytics Engine Configuration and State

The analytics engine configuration and state (AECS) structure contains configuration information used to control the behavior of the various 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).

The Source 2 buffer cannot overlap the Source 1 buffer or the Destination buffer. If Source 2 contains AECS data, then the size of the Source 2 buffer for the purposes of the overlap check is twice the Source 2 Transfer Size. This is true regardless of whether it is being read, written, or both. For example, if Source 2 is being read as AECS and not written, so that it is only referencing the first half, then it is still required that the Source 1 and Destination buffers do not overlap either half of the doubled AECS buffer, even though the second half is not being used.

Depending on the operation, some portions of the nominal AECS may not be relevant and do not need to be read/written. Normally, the AECS is read from the beginning so that the Source 2 Size determines how much of the data at the end of the AECS is omitted.

Additionally, if decompression is enabled and Source 2 is being read as AECS (not written), the Load Partial flag bit in the Decompression Flags can be set (see Section 6.3.2.1). This causes the AECS data that is read to be interpreted as if they were preceded by 448 bytes of 0x00. In other words, the fields in the AECS that fall within the first 448 bytes take their default value of 0, and the fields starting at an offset of 448 are initialized with the data read from Source 2. This can be used for certain decompress operations where the earlier portion of the AECS contains no useful data, and where one is trying to minimize the amount of data read from Source 2 so as to minimize the decompress latency. An example would be when decompressing small pages with “canned” Deflate headers. The headers would be pre-parsed so that the actual compression operation would only want to load the Huffman Tables.

In general, the Source 2 data (for AECS) would contain the AECS data starting at either 0 or 448 and continuing to next multiple of 32 after the last bit of required data. This means that the size of Source 2 will depend on the operation. Some typical sizes (assuming the data starts at an offset of 0) are given in Table 3-2.

Table 3-2 AECS Sizes for Various Operations

Operation	AECS Size
Filter	32
Encrypt/Decrypt	192
Decompress for Indexing	1088 (for AECS Format 1) 1248 (for AECS Format 2)
Decompress	5376
Compress (With Huffman Table)	1568

The amount of AECS data written back to the Source 2 buffer (if writing is enabled) is generally the smaller of the nominal size and the specified size. In other words, it will never write more than the nominal size and it will never write more than the specified size. The nominal sizes are given in Table 3-3.

Table 3-3 Nominal AECS Write Sizes

Condition	Nominal Size
Decompress	5376
Compress with Write AECS Huffman Tables Flag set	1568
Compress without Write AECS Huffman Tables Flag set	64

3.2.1 AECS Format

There are two different formats for the decompress/analytics AECS. This reflects a change in the microarchitecture of the decompression engine after the first generation. There are some format fields in the AECS that indicate which format the data is in. If an AECS in the wrong format is read by the accelerator, the processing terminates with an error.

The appropriate AECS Format is indicated by IAACAP bit 0 (see Section 7.2.2).

This only applies to the AECS for decompress/analytics/crypto. It does not apply to the AECS for compression.

At byte offset 0x1DD there is a 3-bit AECS Format Number. The low order bit in the “Output Bits Valid” field (i.e. at offset 0xB0) is the AECS Format Valid bit. If the Format Valid bit is 0, then the Format Number is reserved.

In Format-1, both of these values must be 0, and in Format-2, both of these values must be 1. See Section 6.4.1.

- If an accelerator expecting a Format-1 AECS reads a Format-2 AECS, it will see an invalid value for the Output Bits Valid field and return an AECS error.
- If an accelerator expecting a Format-2 AECS reads a Format-1 AECS, it will see the incorrect Format Number and return an AECS Format error.

The intent is to provide a mechanism whereby the software library (or application, if not using the library) could check for these errors and, if necessary, convert the AECS to the proper format and resubmit the descriptor. Alternatively, the software could check these bits before submitting the descriptor.

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, where 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 structure contains data used to connect the individual descriptors used to process one logical job. 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 required 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 descriptor (i.e., when write_AECS is “never”).

3.3.2.1 Structure of the Index Table

Each index entry is 8-bytes (64-bits) long. The upper 32-bits is the CRC value of the uncompressed data up to the point corresponding to that index. The lower 32-bits is the bit-offset in the compressed data stream.

It is important to understand what the different indices point to, so that the proper ones can be used for access. In general, the indices for each block point to the boundaries between Deflate Block headers, miniblocks, and the EOB token. In other words, there will be an index entry at the start and at the end of each of these structures within the Deflate stream. The exception is that the hardware will not write an entry corresponding to the start of the first Deflate Block header. It is recommended that software write an initial entry of 0 into the index buffer, set the Destination Address to point to the second index entry, and set the Max Destination Size to the actual size of the index array minus 8.

Note that the AECS field Bit Offset for Indexing is used to pass the bit offset between linked descriptors. If it is desired that the bit offsets in the index table start from a non-zero value, then this field can be written for the first descriptor to set the starting value. In this case, the initial index value written by software would contain this value rather than zero.

To make it easier to map between miniblock number and index entries, it is recommended that the block structure of the Deflate stream be regular. There are two common forms of this. The simplest is where the output consists of a single Deflate block. The index layout for this is illustrated in Figure 3-1.

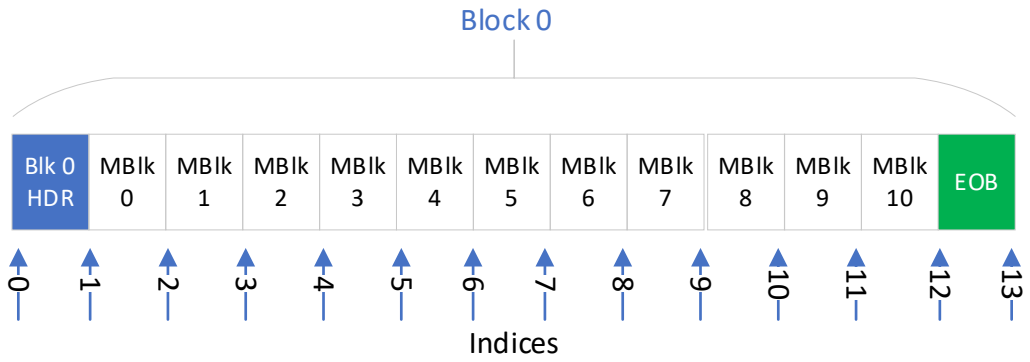

Figure 3-1: Index Arrangement for Single Block Usage

In this example, the first index (created by software) points to the start of the Deflate Block header. Indices 1 through 11 point to the start of the 11 miniblocks. Index 12 points to the start of the EOB token, and Index 13 points to the end of the EOB token, which is also the end of the stream.

The other common structure is where there are multiple blocks, and all of the blocks with the possible exception of the last block are the same size. This is illustrated in Figure 3-2.

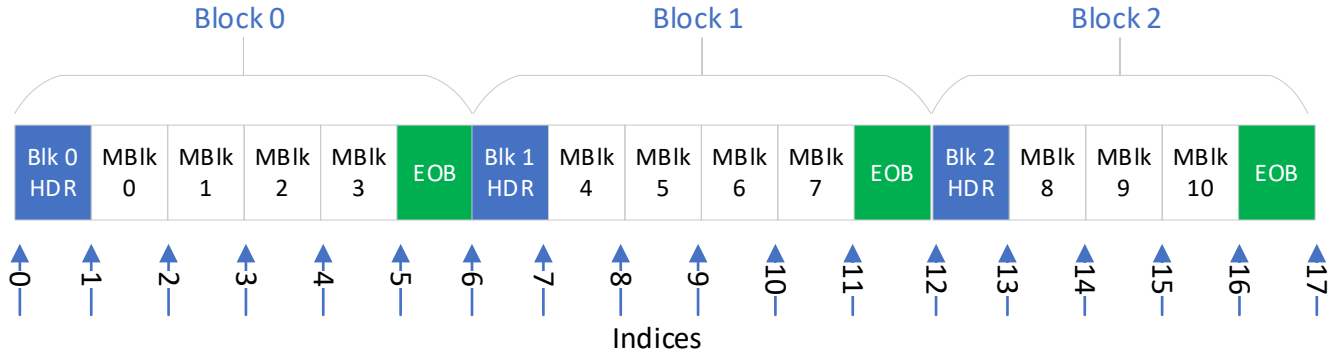

Figure 3-2: Index Arrangement for Multiple Block Usage

In this case, all of the Deflate Blocks except for the last one must have a size that is a multiple of the miniblock size. It is convenient but not required that the size be a power of 2 (e.g., a miniblock sizeof 1kB and a Deflate Block size of 32kB).

Knowing how big the miniblocks are, and how many miniblocks are in each Deflate Block, one can map from an offset in the

uncompressed stream to a particular miniblock, and from there one can compute the index entries that point to the start and end of that Deflate Block's header, and to the start and end of that miniblock.

3.3.3 Decompression With non-Byte-aligned Boundaries

In some applications, a user may want to decompress part of a compressed stream, where the starting and/or ending locations do not align with a byte boundary.

In one formulation, the user has a pointer (actually an offset) to the start of the data to be processed (as a bit offset) and a pointer to the end (actually an offset to first bit not to be processed). If these are called START and END, then the total bit length of the data to be processed would be (END – START). For example, these pointers might have come from the index table (cf. Section 3.3.2.1).

Conceptually, one wants to convert these two pointers to a pointer to the starting byte, the number of bytes to process, and the number of bits to ignore in the first and last byte. This can be done, for example, with the following logic:

```
Start_address = data_buffer_address + START/8;
Size = ((END + 7)/8) – (START/8);
Ignore_start_bits = START & 7;
Ignore_end_bits = 7 & (0 – END);
```

Note that the Size is **not** ((END + 7 – START)/8).

3.3.3.1 Ending on a non-Byte-aligned Boundary

Ending on a non-byte-aligned position is relatively easy. There is a Decompression Flags field "Ignore End Bits" (see Section 6.3.2.1). A non-zero value in this field instructs IAA to stop processing on the last byte the specified number of bits from the end of the byte. For example, a value of 0 (the default) indicates that the whole byte should be processed, whereas a value of 3 would indicate that the first 5 bits should be processed, and the last 3 should be ignored.

The only complication is when the Decompression Bit Order flag is set. In this case, the input consists of an integral number of 16-bit words, or an even number of bytes. In this case, one may need to drop up to 15 bits. This can't always be done with the Ignore End Bits field, because that field is only 3 bits wide. To handle this case, there is 1-bit Ignore End Bits Extension field, which is logically bit-3 of the number of bits to be ignored. That is, in this case the Ignore End Bits Extension field gets bit-3 and Ignore End Bits gets bits 2:0 of the desired value.

This is slightly different for the CRC Flags. In this case, the "Ignore End Bits Extension" is really just bit-3 of the Ignore End Bits field. In other words, for the CRC opcode, the Ignore End Bits field is 4 bits wide, while in the other operations that have Decompress Flags, there is a 1-bit Ignore End Bits Extension field and a 3-bit Ignore End Bits field.

Note that if IAACAP bit-0 is 0, then the Ignore End Bits Extension field is not supported, and CRC, the Ignore End Bits field is only 3 bits wide. In this case, if the Decompress Bit Order flag is set, and the user wants to drop more than 7 bits, this cannot be done.

3.3.3.2 Starting on a non-Byte-aligned Boundary: Method 1

The basic idea here is to store the partial byte (word or DWORD) that starts the data in the AECS input accumulator.

The IAA hardware contains an "input accumulator" which buffers data between coming in from memory and going out to the decoders. Roughly speaking, it takes data in a QWORD at a time, and outputs data at a bit granularity. It is represented in the AECS (cf. Section 6.4.1) as "Input Accumulator Data" and "Size QW *n*". Logically, the data area is considered to be an array of 32 QWORDs, with the number of bits present in each QWORD represented by a separate size field. For example, if the input accumulator was initialized to contain 9 bytes of data, then the data would be written into the first 9 bytes of the Input Accumulator Data field (8 bytes into the first QWORD and 1 byte into the second QWORD), and then Size QW 0 would be set to 64 (64-bits = 8 bytes), and Size QW 1 would be set to 8.

In a typical case, when one is starting on a non-byte-aligned boundary, the input accumulator is empty. Then one can simply write the partial byte (or larger granularity) into the beginning of the input accumulator data (i.e. into the first QWORD of the input accumulator data) and set the Size QW 0 field to the number of bits so written.

Note that the valid data should start at bit-0 of the input accumulator. So, for example, if the data started on bit-3 (i.e., one wanted to drop the first 3 bits), one would set the input accumulator data to (data byte >> 3) and the Size QW 0 field to 5. One would then point the Source 1 Address to the first full byte (i.e., the byte after the partial byte), and adjust the Source 1 Transfer Size accordingly.

This example assumes that Decompress Bit Order is not set. If this flag was set, one would need to write the partial word into the input accumulator rather than the partial byte. Also note that in this case, the bits within each 16-bit word are reversed (e.g., bits 0 and 15 are swapped, bits 1 and 14 swapped, etc.) by the hardware on the way into the input accumulator. So, in this case, if one wanted to drop the first 3 bits of the first word, one would read the partial word, bit-reverse the data, right shift it by 3 bits, write it into the start of the input accumulator data, set the size to 13, and set the Source 1 address to point to the following (full) word.

3.3.3.3 Starting on a non-Byte-aligned Boundary: Method 2

If IAACAP bit 1 (cf. Section 7.2.2) bit is set, then IAA supports an AECS field “Drop Initial Bits” (cf. Sections 6.4.1 and 3.7). In this case, one can just point the Source 1 Address to the partial byte/word and set this field to the desired number of bits to be dropped.

If this IAA device supports decryption, and one wants to do a decrypt-decompress starting on a boundary that is unaligned with a cryptographic block boundary, then one would need to point the Source 1 Address to the start of the cryptographic block and set the Drop Initial Bits field to the number of bits between the start of the cryptographic block and the start of the desired Deflate data. For that reason, the Drop Initial Bits field can be up to 127, which is one bit less than 16 bytes.

For the case of decrypt-decompress, one could use Method 1, but this would require the software to decrypt the initial cryptographic block and then extracting the decrypted data.

3.4 Compression

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

In Huffman-mode, it will read a stream of input bytes, generate a stream of literals and matches, encode them using Huffman tables 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.

Huffman Generation Mode is described in Section 3.4.5.

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. Alternatively, if Huffman Generation is supported, software can use this hardware capability.

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:

Table 3-4 Histogram Table Output in Statistics Mode

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

Note that the compressed data stream generated is a valid Deflate stream. It can be decompressed in the same manner as any other Deflate stream. In addition, any arbitrary miniblock can also be decompressed without decompressing the rest of the stream.

In order 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.

If Compression Indexing is enabled, the input buffer must be a multiple of the miniblock size, except for the last descriptor of a sequence. This is accomplished in some generations with a “Last Descriptor” bit in the Compress AECS (see Section 3.4.7).

The structure of the index table is described in Section 3.3.2.

3.4.4 Compression with a Dictionary

Intel IAA supports Deflate compression with a dictionary. The same dictionary must be used for both compression and decompression. Dictionary compression is most useful when compressing small buffers.

The dictionary itself is just a block of text conceptually prepended to the input stream. The combined dictionary and input stream is compressed, and the compressed tokens associated with the dictionary are dropped. Another way to look at this is that, with dictionary compression, a given bit of data to be compressed can be matched against a location before the start of the buffer.

To compress with a dictionary, the Load Dictionary compression flag must be set, and the dictionary data is placed at the end of normal compression AECS (see Section 6.4.2). I.e., the dictionary data starts in the AECS at byte offset 1568. The dictionary data can be constructed in three different formats with three different sizes. The trade-off is that a larger size for the dictionary data will generally result in a better compression ratio, but it will also cause a longer latency for the compress operation. Some applications may find that the improvement in compression ratio is not worth the increase in compress latency and so opt for a smaller amount of dictionary data.

The dictionary data consists of two variable-length regions. The first is the portion of the dictionary text that is actually being used. The second is a representation of the corresponding hash tables as they would have been created by the hardware. The hash table region can be built with either 2 or 4 pointers per entry. The selection of how big the actual dictionary and hash table entries is called the dictionary "style" and is specified with descriptor flag bits.

The dictionary styles and the corresponding sizes are given in Table 3-5.

Table 3-5 Dictionary Styles and Sizes

Dictionary Style	Size of Dictionary	Size of Hash Table	Total Size
2K Dictionary, 2 Ptrs/Entry	2kB	4kB	6kB
4K Dictionary, 2 Ptrs/Entry	4kB	4kB	8kB
4K Dictionary, 4 Ptrs/Entry	4kB	8kB	12kB

If the raw dictionary is larger than the size of the dictionary as specified by the style, the final bytes of the raw dictionary should be used. If the raw dictionary is smaller, it should be prepended with zero bytes.

3.4.5 Compression Header Generation

Intel IAA has the ability to generate Huffman Tables based on the generated statistics or based on statistics directly input, and to optionally generate a Deflate header corresponding to those tables.

Header Generation can operate in either of two modes: 1-Pass or 2-Pass.

In the 2-Pass mode, dynamic compression with Header Generation is similar to compression without Header Generation. The same input is submitted twice. In the first pass, no compressed output is generated. Instead, the statistics/tables of the output are produced. The second pass actually does the compression.

The difference is that without header generation, the first pass returns the statistics to the software, which must compute the Huffman tables itself. With 2-Pass Header Generation, the first pass returns the Huffman Tables (and optionally the Deflate header), so that the software does not need to perform the Huffman calculations. This both reduces the latency of doing both passes and reduces the CPU load.

The second pass is the same in both cases—it doesn't matter whether the tables and header were created by software or the hardware.

In 1-Pass mode, the compression is done with only one descriptor. In this mode, the hardware will internally do one pass through the data, compute the Huffman Tables, then automatically do a second pass through the data, generating the final compressed output. This is obviously better for the application, but the limitation is that this can only be used if the amount of data to be compressed is no larger than 4kB.

There are three variations of each mode:

- Don't generate a Deflate Header.
- Generate a Deflate Header that is not marked as bFinal.
- Generate a Deflate Header that is marked as bFinal.

In some cases, such as if the input is being compressed in Huffman-only mode with no EOB, it may not be possible to produce a valid Deflate header. Processing such tables should be done without Deflate header generation. If a Deflate header is requested and cannot be created, the descriptor terminates with an error.

Normally, there are no Huffman Codes assigned to tokens with statistics counts of 0. In some cases, the application may want to ensure that all of the tokens are present in the generated tables. This could be used, for example, for "semi-dynamic" compression, where the first portion of a large file is compressed dynamically, and then the same Huffman tables are used for the rest of the file. In this case, all tokens must be represented by Huffman codes, because they might appear later in the file even if they did not in the earlier section.

If the Make Complete Tables bit in the Compress 2 Flags is set, then some of the statistics counters that have 0 values are changed to 1. This means that none of these counters will have a 0 count, and so all tokens will have codes created for them. The counters so changed are the Literal/Length counters up to the last counter that has a non-zero count, or up to counter 272, whichever is greater; and the distance counters up to the last counter that has a non-zero count, or up to counter 23, whichever is greater.

3.4.5.1 2-Pass Header Generation

In 2-Pass Header Generation mode, nothing is written to the destination, and both the destination address and size should be set to 0. Source 2 should be written as AECS. The results are found in the written AECS.

The generated codes are found in the Huffman Tables section of the AECS, as described in Section 6.4.2. The Deflate header is found in the Output Accumulator section. If the Deflate header is not requested, the Output Accumulator section is left unchanged.

Note that because of this, the AECS that is written by the first pass is in the correct format to be read in for the second pass.

In this mode, the Write AECS Huffman Tables flag should be set in the descriptor Compress 2 Flags so that the table portion of the AECS will be written (cf. Section 3.2).

3.4.5.2 1-Pass Header Generation

In 1-Pass mode, there is no restriction on whether Source 2 is read or written. Normally, the Write AECS Huffman Tables flag is not used, as the application would typically not care what the generated Huffman tables were.

If the application does care what tables were used, it can set the Write AECS Huffman Tables flag and have the tables written as part of the AECS. Note, however, that the output accumulator will contain the end of the output bit stream, not the Deflate header.

3.4.5.3 Header Generation with Statistics Input

Normally, the statistics used to construct the Huffman Tables are the counts of the tokens generated by the compressor. If the Header Gen Stats Input bit is set in the Compress Flags, then the normal compress operation is suppressed, and Source 1 and Destination are not used.

Instead, Source 2 must be read and written as AECS. On input, the Huffman Table section (see Section 6.4.2) contains the statistics count values. In each DWORD, bits 18:0 contain the counts, and bits 31:19 should be 0. On output, the AECS contains the generated Huffman Tables and optionally a Deflate header, as is produced with 2-Pass header generation.

In this mode, the Enable Header Generation field must specify one of the 2-Pass modes.

3.4.6 Compression Early Abort

In certain cases, an application may be compressing data that might or might not be compressible to a given level. In such a case, the application might want to abort the compression early if the compression job looks likely to not achieve the desired level of compression.

The advantage of doing this is that if the compression does not achieve the desired level of compression, the application can be notified sooner, and thus the compression latency can be reduced. The disadvantage of doing this is that since the

level of compression is only being estimated, and the estimate will in general be based on only a portion of the input file, then it is possible that a file that actually does achieve the desired level of compression might be considered "incompressible" and aborted, when (in hindsight) it shouldn't have been.

The estimate of the compressed size of the output is the number of literals plus twice the number of references. In other words, it approximates the size of the output by assuming that each literal will take one byte, that each reference will take two bytes, and disregarding the block header.

This feature is controlled by two fields in the Compression 2 Flags (Section 6.3.5.2).

The Early Abort Size field determines when the compressor will perform the check. It can check as soon as 512, 1024, 2048, or 4096 input bytes have been processed. It only checks once, when the specified threshold is passed.

When the check occurs, the size estimate is compared against the input size (as measured at the input of the Deflate compressor) multiplied by the Early Abort Threshold. If the estimate is greater than or equal to the threshold, the compression job is aborted with an error.

Note that the estimated size can never be greater than the input size and can only be equal if no matches were found.

If the input size is equal to the Early Abort Size, then the check is done at the end of the input. This probably only makes sense to do if one is doing Header Generation. In this case, an abort would avoid the header generation and the 2nd half of the processing if any. Otherwise, rather than setting the Early Abort Size to the input size, the application is probably better off setting the Max Destination Size to the desired limit and getting an output overflow if that size is exceeded.

3.4.7 Last Descriptor Bit

In general, a compression job can be continued across multiple descriptors, but there is a case where this cannot be done.

When Compression with Indexing is being done (cf. Section 3.4.3), all of the descriptors require their Source 1 size to be a multiple of the miniblock size, except for the last one. This means that if in this mode a Source 1 size is not a multiple, then this descriptor cannot be continued with a following descriptor.

Starting with Generation 2, this is enforced with a new Compression AECS bit (cf. Section 6.4.2): Last Descriptor Bit. If a compression operation reads an AECS that has this bit set, then the operation is terminated with an error. Conversely, when a compression descriptor cannot be continued with a following descriptor, this bit is set when writing the AECS.

When the Last Descriptor Bit feature is not present, then continuing a compression job when it is not allowed will not result in an error, but it will also not generate correct results. In this case, this restriction must be enforced by software.

The presence of the Last Descriptor Bit is indicated by IAACAP bit 0 (see Section 7.2.2).

3.5 Encryption/Decryption

Intel IAA can perform data encryption. It can also perform data decryption, in which case the decrypted output can be written to the destination buffer, sent to the filter unit, or sent to the decompress unit.

A large encryption/decryption job can be divided into a series of separate descriptors, with the internal state passed between them via the AECS.

Three encryption algorithms are supported: GCM (Galois Counter Mode), AES-CFB (AES Cipher Feedback Mode), and AES-XTS. In each case, supported key sizes are 128 and 256 bits.

Due to space limitations in the descriptor structure, most of the flags and parameters associated with encryption/decryption are contained in the AECS. This is described in Section 6.4.1. Decryption can only be used with the following opcodes: Decrypt, Decompress, CRC64, Scan, or Extract.

AES-XTS cannot be used with indexing.

Decryption can be piped into other operations, so for example, one descriptor can do a decrypt-decompress operation. But encryption cannot be piped, so to do the inverse of the above would take two descriptors: one to do the compression and one to do the encryption.

Cryptographic processing is done one cryptographic block at a time (i.e., 16-bytes at a time). This means that when one crypto job is spread over multiple descriptors, partial blocks are stored in the “Crypto Input Accumulator” within the AECS, and then this data gets processed with the start of the data from the next descriptor. The last descriptor in the job should set the “Flush Crypto Accumulator” flag in the crypto flags in the AECS, so that the final partial block will be processed.

Note that this means the amount of output generated for an Encrypt or Decrypt operation might vary from the input size by up to 31 bytes. For example, on the first descriptor of a multi-descriptor decrypt job, the input could be 31 bytes long, and the output size would be 0. Conversely, on the last descriptor, the input could be 1 byte long, and the output size be 32 bytes.

3.5.1 AES-CFB

For encryption in CFB mode, the Initialization Vector (IV) is encrypted and then XORed with the plaintext to generate the ciphertext. Additionally, the ciphertext becomes the “IV” for the following block. The final ciphertext is returned as the final “IV.”

This same process is done for decryption. The difference is that the ciphertext (which becomes the “IV” of the next block) is an input rather than an output.

3.5.2 GCM

GCM provides both encryption and authentication. The encryption is just AES in Counter Mode. The authentication is provided by computing a cryptographic hash.

In this mode, the hardware only does the processing associated with the bulk data; the rest of the calculations must be done by software. The GCM calculations for encryption are show in Figure 3-3.

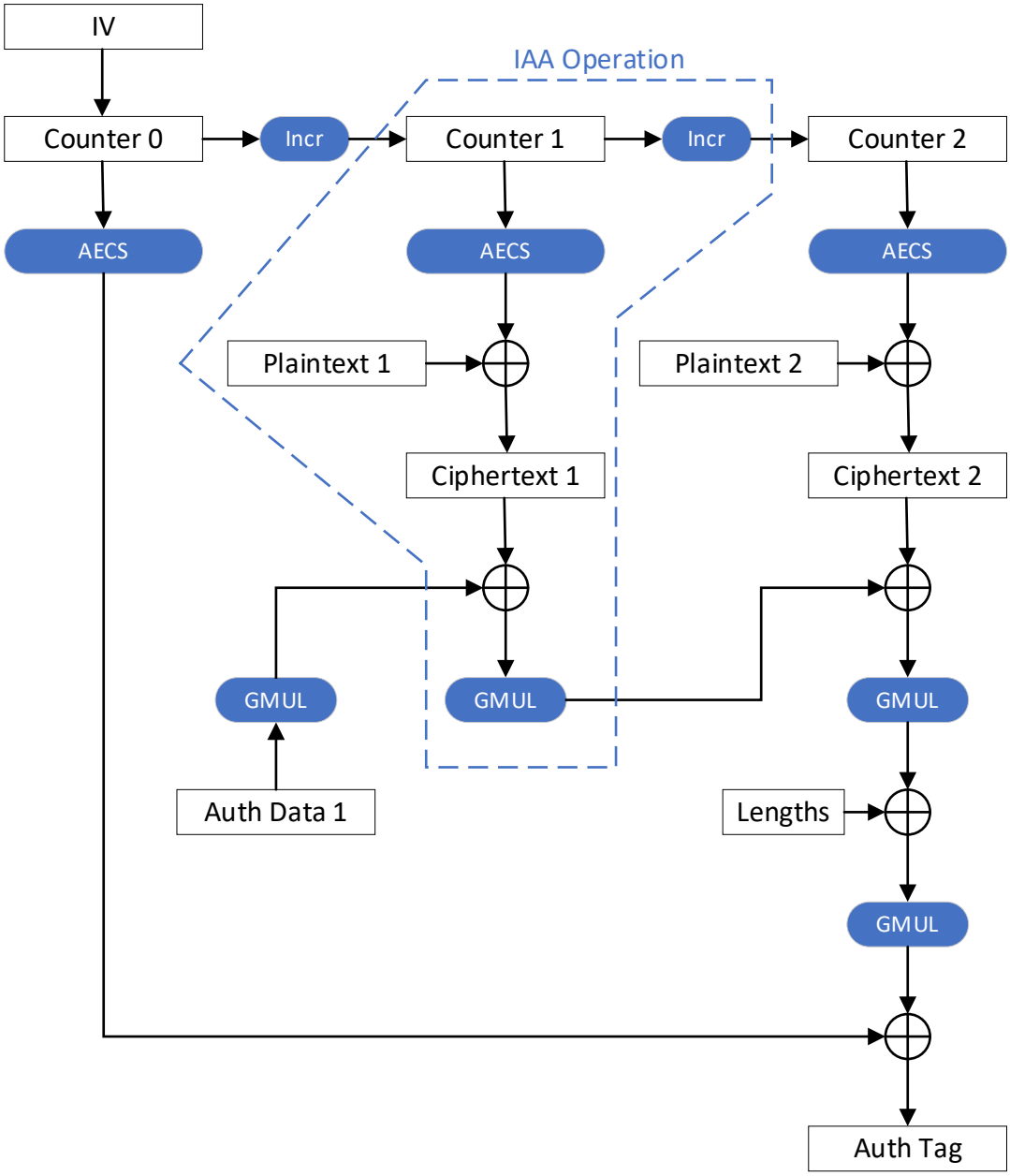

Figure 3-3 GCM Calculations

The dashed section indicates the operations done by the accelerator hardware. In particular, for each block of input data, it will encrypt the IV/counter, XOR that with the input text, XOR the ciphertext with the hash to get a new hash and increment the IV/counter. The initial counter increment, the encryption of Counter 0, and the final addition/hashing of the lengths and encrypted Counter 0 must be done by software. Similarly, the initial hashing of the Additional Authentication Data must be done in software.

3.5.3 XTS

AES-XTS is a block-oriented cipher mode. This means that normally it could only handle buffers that were a multiple of the cipher block size. To allow it to handle arbitrary sized buffers, XTS uses “ciphertext stealing” (CTS) at the end of the stream when the stream size is not a multiple of the cipher block size.

The practical implication of this is that XTS cannot be used on buffers that are smaller than 16 bytes. An attempt to do so will result in an error.

Ciphertext Stealing is invoked at the end of the input stream if the Flush Crypto Accumulator flag is set, and the total number of bytes in this job (i.e., the number of input bytes in this descriptor and in previous linked descriptors) is not a multiple of 16 (i.e., not a multiple of the cipher block size).

3.5.4 Decryption with Indexing

When using index mode to access encrypted data randomly (cf. Section 3.3.2), extra steps must be taken.

In particular, the actual data sent to the accelerator must start at an AES block boundary (16-byte boundary). Then the “Drop Initial Bits” feature (cf. Section 3.7) should be used to skip over the start of generated plaintext, so that the subsequent processing will start at the appropriate location.

Finally, the IV or initial counter value must be calculated appropriately. For CFB, this will generally be the previous block’s ciphertext, except for the first block where it is the IV. For GCM (where only the AES Counter Mode portion is useful), the IV stored in the AECS must be the original IV added to the appropriate number of increments, based on the index of the block starting the decryption.

3.6 Checksum Calculations

As a check, the accelerator generates a pair of checksums of the “raw” data. In particular, there are “compress-like” operations (i.e., Encrypt, Compress) that take original “raw” user data and create a processed version of it. For these functions, the checksums are computed on the original input data. Then there are “decompress-like” operations (i.e., Decrypt, Decompress) that take the processed data and try to recreate the original raw data. For these functions, the checksums are computed on the output data. Finally, there are the filter operations, which take “raw” data and process it further. For these functions, the checksums are computed on the input to the filter processor (i.e., after any decryption or decompression). The basic idea is that the checksums between complementary operations should match. The details of where the checksums are computed is given in Table 3-6.

Table 3-6 Checksum Location

Operation	Type	Checksum Location
Encrypt, Compress	Compress-like	Input
Decrypt, Decompress	Decompress-like	Output
CRC64, Scan, Extract, Select, Expand	Filter	Input to Filter

The accelerator can generate either of two 32-bit CRCs: using the polynomial defined in ITU-T recommendation V.42, or in RFC 3720. It also computes the XOR checksum of the data. For this, the data are treated as 16-bit words. If there are an odd number of bytes, the final byte is zero-extended to 16 bits. Then these 16-bit words are all XORed together.

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.

3.7 Drop Initial Bits vs. Drop Initial Bytes

There are two similar fields found in the AECS (cf. Section 6.4.1): **Drop Initial Bytes** and **Drop Initial Bits**[1].

The Drop Initial Bytes field applies **only** to the input to the filter unit. The Drop Initial Bits field usually applies to the input to the decompress unit. However, if the opcode specifies a filter operation and the decompressor is not enabled, then the Drop Initial Bits field applies to the input of the filter unit.

If both of these fields ("Drop Initial Bits" and "Drop Initial Bytes") are being applied to the input to the filter unit, then at least one must be zero. If both are non-zero, the accelerator will return an error.

The primary use for Drop Initial Bits is to enable indexing on encrypted data (Section 3.5.4). Thus, it would typically be used for the input of the decompressor or filter unit, whichever is immediately downstream from the decryption unit.

Note that if Drop Initial Bits applies to the input of the filter unit, and the filter parser is PRLE, then Drop Initial Bits must be 0.

For a filter operation without decompression, the Source 1 size must be greater than the amount of data being dropped, except for EXPAND, where the Source 1 size must be greater or equal to the amount of data being dropped.

If decompression is enabled without decryption, then the sum of the Drop Initial Bits + Ignore End Bits cannot be greater than the Source 1 Size. If decompression is enabled with decryption, and Flush Crypto Input Accumulator is set, then the sum cannot be greater than the Source 1 Size plus the size of the Crypto Input Accumulator.

3.8 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 or decryption. The primary input is parsed as described in 3.8.1. 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 filter operation that uses the secondary input uses default values for any configuration information that would have been read from the AECS. This means that such operations cannot be used with decryption.

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 must resubmit the descriptor with a larger output buffer.

3.8.1 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.

3.8.1.1 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).

[1] Drop Initial Bits is not available on all IAA implementations. The availability of this feature is indicated by IAACAP bit 1 (see Section 7.2.2).

3.8.1.2 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)
```

3.8.2 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 output: 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 scan query, i.e., “is an element within a given range?,” generate a bit vector, where each bit represents membership in that range. 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.

An optional flag bit forces the output to be considered an array, even if the bit-width is 1. This can be used to unpack a bit vector into an array of bytes, words, or DWORDs. Use of the Force Array Output Mod flag does not change whether one can apply the Invert Output flag. That is, if a nominal bit-vector output is forced to use array output modification, it can still be inverted.

3.8.2.1 Modification When Output is Normally a Bit Vector

If the output of a 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 the index of any output element is too large for the specified output width, the operation stops and reports an error.

3.8.2.2 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 3.8.2.1 apply, unless the Force Array Output flag is set.

3.8.3 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 232 of the output values).
- Minimum value.
- Maximum value.

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

Note that if the Force Array Output flag is set, then the array aggregates are computed. So, in most cases, the min and max values would be 0 and 1, rather than the indices of the first and last 1-bit.

3.9 Chaining of Functions

- If IAACAP bit 2 is 0, the output of decryption/decompression can be chained into the input of the filter unit for all operations other than CRC64.
- If IAACAP bit 2 is 1, the output of decryption/decompression can be chained into the input of the filter unit for all operations.

3.10 Operation Types

The operations No-op, Drain, and Translation Fetch are the same as in Intel® DSA.

3.10.1 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 control aspects of the decompression operation. The “Enable Decompression” flag must be set.

Optionally the input to the decompressor can be taken from the output of the decryption unit.

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.

3.10.2 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 Flags and Compression Flags 2 control aspects of the compression operation.

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

In one usage, the output buffer is sized large enough to store the input as a “stored block.” If the compressed output is too large to fit into this buffer, then the partial results of the compression should be thrown away by software and replaced by a stored block.

Usage of the Source 2 / AECS is:

Condition	Source 2 Read	Source 2 Write
Statistics Mode w/o dictionary	Not allowed	Not allowed
Statistics Mode with dictionary	Required	Not allowed
1-Pass Header Gen Mode, w/o dictionary	Optional	Optional
1-Pass Header Gen Mode, with dictionary	Required	Optional
2-Pass Header Gen Mode	Optional	Required
Otherwise	Required	Optional

3.10.3 CRC64

The CRC64 operation computes an arbitrary CRC up to 64-bits in width.

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.

The following are some well-known CRCs and the programming required to generate them:

Table 3-7 Examples of CRC64 Parameters

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	0000000000000XXXX

3.10.4 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.

3.10.5 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 3.8.2.

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).

3.10.6 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 3.8.2.

3.10.7 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 3.8.2.

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 through Table 4-3 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 Translation Fetch.
- 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. If Read Source 2 is 1, or if Write Source 2 is non-zero, then Source 2 points to an AECS structure (cf. Section 3.2), and the size used for the overlap check is twice the Source 2 Transfer Size.

These checks may be performed in any order. Thus, an indication of one type of error in the completion record does not imply that there are not also other errors. The same invalid descriptor may report different error codes at different times or with different versions of the device.

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 and a more detailed view of flag restrictions, see Section 6.

Table 4-1 shows what Operation Flags are allowed for each Intel IAA-specific operation. Table 4-2 shows the decompress and filter flags that are allowed for the analytic and crypto operations. Note that the filter flags are not present in descriptors for encrypt and decrypt, so those cells are X-ed out. The compression flags and the CRC flags are only defined for a single opcode, so these are not listed here. Table 4-3 summarizes the descriptor fields that are allowed for each Intel IAA-specific operation type. For common operations, refer to the Intel DSA specification.

Operation Flag bits 23, 7:6, and 0 are reserved for all operation types. For the cases of Compression Flags, Encryption Flags, and CRC64 Flags, all of the non-reserved fields are allowed for all operations for which those flags are defined, so they are not listed in tabular form here.

Table 4-4 gives additional constraints on the input size.

Additional operation-specific reserved fields and flags are described with the respective descriptor details in Section 6.

Table 4-1 Operations Flags Applicability

		Decrypt	Encrypt	Decompress	Compress	CRC64	Scan	Extract	Select	Expand
Op Flags	Block on Fault	●	●	●	●	●	●	●	●	●
	Comp Rec Addr Valid	●	●	●	●	●	●	●	●	●
	Req Comp Record	●	●	●	●	●	●	●	●	●
	Req Comp Interrupt	●	●	●	●	●	●	●	●	●
	Completion Record TC Selector	●	●	●	●	●	●	●	●	●
	Source 1 TC Selector	●	●	●	●	●	●	●	●	●
	Dest TC Selector	●	●	●	●		●	●	●	●
	Cache Control	●	●	●	●		●	●	●	●
	Strict Ordering	●	●	●	●		●	●	●	●
	Dest Readback	●	●	●	●		●	●	●	●
	Read Source 2	●	●	●	●	●	●	●	●	●
	Write Source 2	●	●	●	●					
	Source 2 TC Selector	●	●	●	●	●	●	●	●	●
	CRC Select	●	●	●	●	●	●	●	●	●
	AECS R/W Toggle Selector	●	●	●	●	●	●	●		

Table 4-2 Analytics Flags Applicability

		Decrypt	Encrypt	Scan	Extract	Select	Expand
Decompress	0: Enable Decompress			●	●	●	●
	1: Flush Output	●	●	●	●	●	●
	2: Stop on EOB			●	●	●	●
	3: Check for EOB			●	●	●	●
	4: Select bFinal EOB			●	●	●	●
	5: Decompress Bit Order			●	●	●	●
	8-6: Ignore End Bits			●	●	●	●
	9: Suppress Output	●	●	●	●	●	●
	13: Load Partial			●	●		
	14: Ignore End Bits Extension			●	●	●	●
Filter	1-0: Source 1 Parser			●	●	●	●
	6-2: Source 1 Width			●	●	●	●
	11-7: Source 2 Width						
	12: Source 2 Bit Order					●	●
	14-13: Output Width			●	●	●	●
	15: Output Bit Order			●	●	●	●
	16: Invert Output			●	●	●	●
	27: Force Array Output Mod			●	●	●	●

Table 4-3 Operation-Specific Allowed Fields

			Decrypt	Encrypt	Decompress	Compress	CRC64	Scan	Extract	Select	Expand
Bytes	4-6	Operations Flags	●	●	●	●	●	●	●	●	●
	7	Operation	●	●	●	●	●	●	●	●	●
	8-15	Comp Rec Address	●	●	●	●	●	●	●	●	●
	16-23	Readback Address 1									
		Source 1 Address	●	●	●	●	●	●	●	●	●
	24-31	Readback Address 2									
		Destination Address	●	●	●	●		●	●	●	●
	32-35	Source 1 Size	●	●	●	●	●	●	●	●	●
	36-37	Comp Interrupt Handle	●	●	●	●	●	●	●	●	●
	38-39	Decompress Flags			●			●	●	●	●
		Encryption Flags	●	●							
		Compression Flags				●					
		CRC Flags					●				
	40-47	Source 2 Address	●	●	●	●	●	●	●	●	●
	48-51	Destination Size	●	●	●	●		●	●	●	●
	52-55	Source 2 Size	●	●	●	●	●	●	●	●	●
	56-59	Filter Flags						●	●	●	●
		Compression 2 Flags				●					
		CRC64 Polynomial					●				
	60-63	Num Elements						●	●	●	●
		CRC64 Polynomial					●				

Table 4-4 Data Size Checks

Field/Data Item	Restriction
Source 1 Size	For a filter operation without decompression, the Source 1 Size must be greater than the amount of data being dropped. The exception is for EXPAND, where the Source 1 Size must be greater or equal to the amount of data being dropped.

4.3 AECS Checks

Table 4-5 and Table 4-6 give constraints on AECS parameters. See Section 6 for additional restrictions and details.

Table 4-5 Non-Compress AECS Checks

	Field/Data Item	Restriction
Format 1	Output Bits Valid	Bit 0 of Output Bits Valid must be 0. See Section 3.2.1.
Format 2	Output Bits Valid	Bit 0 of Output Bits Valid must be 1. See Section 3.2.1.
	AECS Format	AECS format must be 1. See Section 3.2.1.
	Output Bits Valid	Output Bits Valid (ignoring bit 0) must be a multiple of 8.
	Drop Initial Bits / Drop Initial Bytes	For a filter operation without decompression, at least one of Drop Initial Bits and Drop Initial Bytes must be 0.
	Drop Initial Bits / Drop Initial Bytes	If the opcode is Encrypt or Decrypt, both Drop Initial Bits and Drop Initial Bytes must be 0.
	Input Accumulator Size	None of the Input Accumulator QW Sizes can be greater than 64.
	Input Accumulator Size	If any of the Input Accumulator QW Sizes is 0, all higher order sizes must be 0; i.e., the non-zero sizes must be in a contiguous group starting at index 0.
	Input Accumulator Size	For the Input Accumulator: If (size[31] ≠ 0) then sum (size[0] + … +size[31]) must be ≥ 193). If (size[30] ≠ 0) then sum (size[0] + … +size[30]) must be ≥ 129).
	Source 2 Size	If the History Buffer Write Pointer is non-zero, that portion of the history buffer must be completely read; the number of bytes specified by Source 2 Size cannot end in the middle of the specified portion of the History Buffer
	History Buffer Write Pointer / Output Bits Valid	If Indexing is enabled, then (History Buffer Write Pointer{2:0}) must be the same as (Output Bits Valid{5:3})
	Drop Initial Bits	If Decompress is not enabled and the parser is PRLE, Drop Initial Bits must be 0.
	Drop Initial Bits	If the opcode is CRC64 and decompress is not enabled, Drop Initial Bits must be 0.
	Drop Initial Bits / Ignore End Bits	If Decompression is enabled and Decryption is not, the sum of Drop Initial Bits and Ignore End Bits cannot be greater than the Source 1 Size.
	Drop Initial Bits / Ignore End Bits	If Decompression and Decryption are both enabled and Flush Crytpo Input Accumulator is set, the sum of Drop Initial Bits and Ignore End Bits cannot be greater than the Source 1 Size plus the Crypto Input Accumulator Size.
	Ignore End Bits	If Decrypt is enabled and Flush Crypto Input Accumulator is not set, Drop End Bits must be 0.
	Stored Block Bytes Remaining	If the Decompress State is 2 or 6 (Looking at a Stored Block), the Stored Block Bytes Remaining must be non-zero.
	Cipher Flags	If the opcode is Encrypt or Decrypt, Enable Crypto must be set.
	Cipher Flags	Enable Crypto can only be used for opcodes: Encrypt, Decrypt, Decompress, CRC64, Scan, and Extract.
	Cipher Flags	Any operation other than Encrypt, Decrypt, and Decompress that has Enable Crypto set must also set Flush Crypto Accumulator.
	Cipher Accumulator Size	Any operation other than Encrypt, Decrypt, and Decompress that has Enable Crypto set must set Cipher Accumulator Size to 0.
	Cipher Accumulator Size	The Cipher Accumulator Size must be no greater than 40.
	Cipher Algorithm	The Cipher Algorithm must be 0, 1, or 2.
	History Buffer Write Pointer	If the low 15 bits of the History Buffer Write Pointer ≥ 4096, then bit 15 must be set.

Table 4-6 Compress AECS Checks

Field/Data Item	Restriction
Output Accumulator Size	The Output Accumulator Size must be ≤ 64 × 32.
Output Accumulator Size	If Enable Header Generation is not 0, then the Output Accumulator Size must be < 64 × 32.

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 instead written to the SWERROR register or to the Event Log if enabled.

The operation status codes are the same as for Intel DSA, with the exception of those listed in Table 4-7.

Table 4-7 Operation Status Codes

Status Code	Description
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 Operation Flags. A field in Operation Flags contains an unsupported or reserved value.
0x12	Non-zero reserved field (other than a flag).
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, size, or stride field: • The AECS address was not a multiple of 32 bytes. • The AECS size was not a multiple of 32 bytes. • In a Translation Fetch operation: Region Stride is less than 4096 or is not a power of 2
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 flag in bytes 38:39 of the descriptor. Depending on the opcode, this could be Decompression, Compression, Encryption, or CRC Flags. A field in the flags contains an unsupported or reserved value.
0x31	Invalid flag in bytes 56:59 of the descriptor. Depending on the opcode, this could be Filter or Compression 2 Flags. A field in the flags contains an unsupported or reserved value.
0x32	Invalid Input Size. The input size when the Decompress Bit Order flag is set was not a multiple of 2.
0x33	Invalid Number of Elements: Number of Elements is 0 for a filter operation.
0x35	Invalid Invert Output: The Invert Output flag was used when the output was not a bit-vector.

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 4-8:

Table 4-8 Error Codes

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 LL 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 2N-1 or last code is greater than 27.
0x09	Bad LL code lengths	First code of length N is greater than 2N-1 or last code is greater than 215.
0x0A	Bad DIST code lengths	First code of length N is greater than 2N-1 or last code is greater than 215.
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 Miniblocks	During index generation as part of decompress, a match referenced data in a different miniblock.
0x1B	Length Spans Miniblocks	During index generation as part of decompress, a match had a length extending into the next miniblock.
0x1C	Invalid Block Size	During index generation as part of decompress, a block header occurred that was not on a multiple of the miniblock size.
0x1D	Verify Failure	Internal verification hardware detected a possible error in the output.
0x1E	Invalid Huffman Code	A compression job tried to use a Huffman code with zero size.

Error Code	Detected Error	Description
0x1F	PRLE bit-width Invalid	The bit-width specified in the first byte of a PRLE stream was greater than 32, was equal to 0, or was missing.
0x20	Too Few Elements Processed	The input stream ended before specified Number of Input Element was seen.
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.
0x25	Administrative Timeout	This operation was terminated because it exceeded a temporary timeout limit imposed by an administrative command. Resubmitting the descriptor is recommended.
0x26	Invalid Crypto Flag	Crypto was enabled for an opcode that doesn't support cryptography, or some cryptographic flag/parameter was used inappropriately or had an invalid value.
0x27	Invalid Crypto Size	A crypto operation was attempted with the XTS algorithm, with a total input size less than 16 bytes.
0x28	Data Size Too Large	A compress descriptor with 1-pass Header Generation enabled attempted to compress more than 4kB.
0x29	Compression Early Abort	The Compression Early Abort feature was triggered. See Section 3.4.6.
0x2A	Can't Make Deflate Header	A Deflate header couldn't be generated because there were not at least 257 Literal/Length tokens defined.
0x2B	Invalid Compression Linking	A compression descriptor with indexing enabled was attempted after the previous linked descriptor compressed an amount of data that was not a multiple of the miniblock size. See Section 3.4.7.
0x2C	Deflate Header Too Large	The generated Deflate header could not fit into the output accumulator.
0x2D	AECS Format Error	The AECS format fields contained incorrect values. See Section 3.2.1.
0x2E	Inconsistent State	An inconsistency in the internal state of the decompress engine was noticed. This typically is indicative of a corrupted AECS image being loaded.

5 SOFTWARE ARCHITECTURE

5.1 Intel® Query Processing Library

The Intel® Query Processing Library (QPL) provides user-mode access to the device in a manner that is more user-friendly and less dependent on the hardware interface. 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.

It also provides optimized software-only implementations, for cases where the accelerator hardware is not present.

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 generic Intel IAA 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	Reserved	PASID		0
Completion Record Address								8
Source 1 Address								16
Destination Address								24
Flags1		Completion Interrupt Handle		Source 1 Transfer Size				32
Source 2 Address								40
Source 2 Transfer Size				Maximum Destination Size				48
Flags3				Flags2				56

Figure 6-1 Generic Intel® IAA Descriptor Format

The details of the descriptors for various operations vary. In particular, some of the fields may be reserved, and the details of the "Flags" fields will be operation specific. This section describes the common fields, and then the following sections will describe the variations specific to different opcodes.

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.

Table 6-1 Descriptor Trusted Fields

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.

6.1.2 Operation

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

This field specifies the operation to be executed.

Table 6-2 Operation Types

Operation Code	Type
0x00	No-op
0x01	Unused
0x02	Drain
0x03 – 0x09	Reserved
0x0A	Translation Fetch
0x0B-0x3F	Reserved
0x40	Decrypt
0x41	Encrypt
0x42	Decompress
0x43	Compress
0x44	CRC64
0x45-0x4F	Reserved
0x50	Scan
0x51	Reserved
0x52	Extract
0x53	Select
0x54	Reserved
0x55	Reserved
0x56	Expand

6.1.3 Operation Flags

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

See the Intel® DSA architecture documentation for meanings and restrictions for these flags for No-op, Drain, and Translation Fetch.

Table 6-3 Operations Flags

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 the CRC64 opcode when IAACAP bit 0 is 0.
20	Reserved: Must be 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. 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 for operation types other than Decompress, Compress, Encrypt, and Decrypt. This field is reserved if the operation is Compress and in the Compression Flags: Stats Mode is 1. A value of 1 is required if the Compression Flag Enable Header Gen has a value of 5, 6, or 7.

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.
	The value 2 is required for Select and Expand. The value 2 is reserved for all other operation types.
	A value of 1 is required when encryption/decryption is enabled. (See Section 6.4.1)
	This field is reserved for No-op, Drain, and Translation Fetch. If IAACAP Bit-0 is 0, then this field is reserved for CRC64.
	If the operation is Compress:
	• This field is reserved if in the Compression Flags: Stats Mode is 1 and Load Dictionary is 0
	• This field must be 1 if in the Compression Flags any of the following conditions are true:
	  ○ Load Dictionary is not 0
	  ○ Stats Mode and Enable Header Generation are both 0
	  ○ Header Gen Stats Input is set in the Compression Flags

15	Reserved: Must be 0.
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, Drain, Translation Fetch, and CRC64.

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, so they become globally observable in the exact order issued by the device. This field is reserved for operation types that do not write to memory: No-op, Drain, Translation Fetch, and CRC64. 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 and Translation Fetch.
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 is reserved when Readback Address 2 Valid is 0. This field is reserved for operation types that do not use Destination Address: No-op, Drain, Translation Fetch, and CRC64.
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 is reserved when Readback Address 1 Valid is 0. This field is reserved for operation types that do not use Source 1: No-op and Drain.
8	**Cache Control** For operations that write to memory: **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 field in GENCAP is 0, this field is reserved. This field is reserved for No-op, Drain, Translation Fetch, and CRC64.
7:5	Reserved: Must be 0.

Operation Flags	
Bits	**Description**
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 (See the Intel DSA architecture specification for further details). For Drain, either Request Completion Record or Request Completion Interrupt must be set.
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 Translation Fetch. This flag must be 0 if Completion Record Address Valid is 0. For Drain, either Request Completion Record or Request Completion Interrupt must be set.
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 Translation Fetch.
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, Descriptor List Address, or Drain Readback Address, all of which always block on fault. See the Intel DSA architecture specification for further details. 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.

6.1.4 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 Translation Fetch.

6.1.5 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.

6.1.6 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 Maximum Destination Size.

This field is reserved for No-op, Translation Fetch, and CRC64.

This field is ignored if Maximum Destination Size is zero.

6.1.7 Source 1 Transfer Size

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

This field indicates the number of bytes to be read from the Source1 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 operation Compress, if Enable Indexing is not 0, the size must be a multiple of specified miniblock size, unless it is the last descriptor in the sequence. See Section 3.4.7 for further information.

If the operation is other than No-op or Drain, then at least one of Source 1 Transfer Size, Source 2 Transfer Size, and Maximum Destination Size (or in some generations Source 1 Transfer Size, and Maximum Destination Size) must be non-zero. Which set of values is considered in indicated by IAACAP bit 0 (see Section 7.2.2).

Most of the functions require some Source 1 data. In this context, no Source 1 data means that either the Source 1 Transfer Size is zero, or that it is equal to the dropped bits or dropped bytes as appropriate. The only functions (that process data) that accept no Source 1 data are: Decrypt, Encrypt, Decompress, Compress, and Expand. The other data processing operations (e.g., CRC64, etc.) require a non-zero amount of Source 1 data.

6.1.8 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.

6.1.9 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.

The Source 2 buffer must not overlap the Source 1 buffer or the Destination buffer. If Source 2 points to an AECS, then the size used for the overlap check is twice the Source 2 Transfer Size.

This field is reserved if the operation type is No-op, Drain, or Translation Fetch or if Read Source 2 and Write Source 2 are both 0.

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

6.1.10 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, or CRC64.

If the operation is other than No-op or Drain, then one of Source 1 Transfer Size, Source 2 Transfer Size, and Maximum Destination Size must be non-zero.

For Translation Fetch, this field contains the Region Stride.

6.1.11 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, and it must be no greater than 64kB. 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 or Drain, or if Read Source 2 and Write Source 2 are both 0.

If the operation is Select or Expand, then the Source 2 Transfer Size must be non-zero.

If the operation is Compress, and Source 2 is being read, then the Source 2 transfer size must be 1,568 bytes plus the size of the dictionary data, if any. The dictionary data size is determined by bits 11:10 of the Compression Flags (Load Dictionary). In this case, the size of the Source 2 transfer size must be equal to the value from Table 6-4.

Table 6-4 Source 2 Sizes for Different Values of Load Dictionary

Load Dictionary	Src2 Transfer Size
0	1,568
1	7,712
2	9,760
3	13,856

If the operation is Encrypt or Decrypt, then the Source 2 transfer size must be 192. For other operations which have decrypt enabled, the size must be at least 192.

If the AECS is written, then the amount of data written may be less than the specified size. See Section 3.2 for more details.

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. Software should not depend on the value of unused fields (including fields that are unused for specific operation types). The completion record address must be 64-byte aligned.

Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0	bytes
Bytes Completed				Unused	Fault Info	Error code	Status	0
Fault Address								8
Unused				Invalid Flags				16
XOR Checksum		Unused	Output Bits	Output Size				24
Min / First				CRC				32
Sum / Population count				Max / Last				40
Crypto Hash Low								48
Crypto Hash High								56

Figure 6-2 Intel® IAA Completion Record Format

For some operations, some of these fields are unused and are written as 0. For example, the Min/First, Max/Last, and Sum/Population Count fields are only relevant for filter operations. Likewise, the Crypto Hash fields are only relevant for operations doing encryption or decryption.

A slight variant is generated for the CRC64 operation. The completion record for that will be described in Section 6.3.6.

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.

Table 6-5 Completion Record Status Field

Bits	Description
7	**R/W** (Not used unless Operation Status indicates a translation fault – code 0x03, 0x04 or 0x1a) **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 this field.

6.2.2 Error Code

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

When the Status field is equal to 0x0A, 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 Fault Info

Offset: 2; Size: 1 byte (8 bits)

If the operation was partially completed due to a page fault and Completion Record Fault Info Support in GENCAP is 1, this field contains additional information about the fault encountered.

Table 6-6 Completion Record Fault Info

Bits	Description
7:4	Unused.
3:1	**Operand Identifier:** **0**: Unknown **1**: Source 1 **2**: Source 2 (fault detected when reading Source 2) **3**: Destination **4**: Source 2 (fault detected when writing Source 2) **5**: Completion Record Address **6-7**: Reserved
0	**Fault Address Masked** **0**: The fault address field contains the address that caused the fault. **1**: The fault address is masked or not available

6.2.4 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 the 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.
- Otherwise (e.g., if the operation fully completed or terminated due to some other error), this field contains 0.

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 Translation Fetch operation that resulted in a page fault.

6.2.5 Fault Address

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

If the operation terminated due to a page fault and Completion Record Fault Info Support in GENCAP is 1, the Fault Info field specifies if the Fault Address is available. If Completion Record Fault Info Support in GENCAP is 0, this field always contains the address that caused the fault.

6.2.6 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.

6.2.7 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 Translation Fetch.

6.2.8 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 Translation Fetch.

In the case of compress, if the Compress Bit Order flag is set (so that the output consists of 16-bit words), then Output Bits gives the number of bits written in the last word (0 if all were written).

6.2.9 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. See Section 3.6 for further details. This field is not used for the following operation types: No-op, Drain, or Translation Fetch.

6.2.10 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). See Section 3.6 for further details. This field is not used for the following operation types: No-op, Drain, or Translation Fetch.

6.2.11 Aggregates

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

These fields contain information about the output, as follows:

Table 6-7 Completion Record Aggregates fields

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.

These fields are only used for the following operation types: Scan, Extract, Select, and Expand.

6.2.12 Crypto Hash

Offset: 48; Size: 16 bytes (128 bits)

These fields contain the generated hash value when the operation involves encryption or decryption, and the algorithm is GCM. Otherwise, they are written as 0.

6.3 Descriptor Types

6.3.1 Intel® DSA Operations

The following operations are imported from Intel® DSA:

- No-op (0x00)
- Drain (0x02)
- Translation Fetch (0x0A)

These behave the same as in Intel DSA.

For further information about these operations, see the Intel DSA architecture documentation.

6.3.2 Decompress Descriptor (0x42)

This descriptor applies to the Decompress operation.

Decompress Descriptor

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
Destination Address								24
Decompression Flags		Completion Interrupt Handle		Source 1 Transfer Size				32
Source 2 Address								40
Source 2 Transfer Size				Maximum Destination Size				48
Reserved								56

Figure 6-3 Decompress Descriptor

The Decompression Flags are described in Table 6-8.

The flags enabling and controlling Decryption are located in the AECS (see Table 6-15).

6.3.2.1 Decompression Flags

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

Table 6-8 Decompression Flags

Decompression Flags	
Bits	**Description**
15	Reserved: Must be 0.
14	**Ignore End Bits Extension** This provides a fourth high-order bit for Ignore End Bits. • This bit is reserved unless Decompress Bit Order is set. • If Decompress Bit Order is set, then the number of ignored bits in the last 16-bit word is given by {Ignore End Bits Extension, Ignore End Bits}.
13	**Load Partial** **0:** Load AECS starting at offset 0. **1:** Load AECS starting at offset 448. This field is reserved except when decompress is enabled and Read Source 2 has the value 1 (Read as AECS). See Section 3.2. This field is reserved if Write Source 2 is not 0.

Decompression Flags	
Bits	**Description**
12:10	**Enable Indexing** **0:** Indexing is not enabled. **1:** Enable indexing with a miniblock size of 512 bytes. **2:** Enable indexing with a miniblock size of 1kB. **3:** Enable indexing with a miniblock size of 2kB. **4:** Enable indexing with a miniblock size of 4kB. **5:** Enable indexing with a miniblock size of 8kB. **6:** Enable indexing with a miniblock size of 16kB. **7:** Enable indexing with a miniblock 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.
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. • If decryption is enabled, then this flag must be 0 unless Flush Crypto Input Accumulator is set. • If Decompress Bit Order is set, then {Ignore End Bits Extension, Ignore End Bits} gives the number of bits to ignore in the last 16-bit word.
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.) If this flag is set, then the Source 1 Size must be an even number of bytes.
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.
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 data or decrypted data directly to the filter function. **1:** Decompress Source 1 data or decrypted data. • 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.

6.3.3 Analytics Descriptor (0x50, 0x52, 0x53, 0x56)

This descriptor applies to Scan, Extract, Select, and Expand, although some of the fields are not used for some of the operations.

Analytics Descriptor

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
Destination Address								24
Decompression 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 6-4 Analytics Descriptor

The Decompression Flags are described in Table 6-9, and the Filter Flags in Table 6-10.

The flags enabling and controlling Decryption are located in the AECS (see Table 6-15).

6.3.3.1 Decompression Flags

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

Table 6-9 Decompression Flags

Decompression Flags	
Bits	**Description**
15	Reserved: Must be 0.
14	**Ignore End Bits Extension** This provides a fourth high-order bit for Ignore End Bits. • This bit is reserved unless Decompress Bit Order is set. • If Decompress Bit Order is set, then the number of ignored bits in the last 16-bit word is given by {Ignore End Bits Extension, Ignore End Bits}.
13	**Load Partial** **0:** Load AECS starting at offset 0. **1:** Load AECS starting at offset 448. This field is reserved except when decompress is enabled and Read Source 2 has the value 1 (Read as AECS). See Section 3.2. This field is reserved if Write Source 2 is not 0.
12:10	Reserved: Must be 0.
9	**Suppress Output** **0:** Decompressed / filter data is written to the output stream. **1:** Decompressed / filter data is not written to the output stream.
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. • If decryption is enabled, then this flag must be 0 unless Flush Crypto Input Accumulator is set. • If Decompress Bit Order is set, then {Ignore End Bits Extension, Ignore End Bits} gives the number of bits to ignore in the last 16-bit word.

Decompression Flags	
Bits	**Description**
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.) If this flag is set, then the Source 1 Size must be an even number of bytes.
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.
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 data or decrypted data directly to the filter function. **1:** Decompress Source 1 data or decrypted data. • 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.

6.3.3.2 Filter Flags

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

Table 6-10 Filter Flags

Filter Flags	
Bits	**Description**
31:28	**Reserved:** Must be 0.
27	**Force Array Output Modification** **0:** Treat nominal 1-bit output as a bit-vector for output modification. **1:** Treat nominal 1-bit output as an array for output modification. This flag is ignored if the nominal output bit-width is not 1. For more details, see Section 3.8.2. This field is reserved unless Output Width has a non-zero value. This field does not affect whether Invert Output is allowed.
26:17	**Reserved:** Must be 0.
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 reserved for all current 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. 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 Section 3.8.1.2. **3:** Reserved. If this field is 2, then Drop Initial Bits must be 0. (See Section 3.7)

6.3.3.3 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.

This field is reserved for all operations except for: Scan, Extract, Select, and Expand.

6.3.4 Decrypt/Encrypt Descriptor (0x40, 0x41)

This descriptor is used for the Encrypt and Decrypt operation.

Decrypt/Encrypt Descriptor

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
Destination Address								24
Cipher Flags		Completion Interrupt Handle		Source 1 Transfer Size				32
Source 2 Address								40
Source 2 Transfer Size				Maximum Destination Size				48
Reserved								56

Figure 6-5 Decrypt/Encrypt Descriptor

The flags enabling and controlling Encryption/Decryption are located in the AECS (see Table 6-15). The Cipher Flags are described in Table 6-11.

6.3.4.1 Cipher Flags

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

Table 6-11 Cipher Flags

Cipher Flags	
Bits	**Description**
15:10	**Reserved:** Must be 0.
9	**Suppress Output** **0:** Encrypted data is written to the output stream **1:** Encrypted data is not written to the output stream.
8:2	**Reserved:** Must be 0.
1	**Flush Output** **0:** A partial output word is saved in the AECS. This value should be used for an Encrypt 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, the operation returns an error. This value should be used for an Encrypt descriptor that is the last (or only) descriptor in a job.
0	**Reserved:** Must be 0.

6.3.5 Compress Descriptor (0x43)

This descriptor is used for the Compress operation.

Compress Descriptor

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
Destination Address								24
Compression Flags		Completion Interrupt Handle		Source 1 Transfer Size				32
Source 2 Address								40
Source 2 Transfer Size				Maximum Destination Size				48
Reserved				Compression 2 Flags				56

Figure 6-6 Compress Descriptor

The Compression Flags are described in Table 6-12. The Compression 2 Flags are described in Table 6-13.

6.3.5.1 Compression Flags

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

Table 6-12 Compression Flags

Compression Flags	
Bits	**Description**
15	**Header Gen Stats Input** **0**: Input is data to be compressed **1:** Input contains statistics to be processed through Header Generation. See Section 3.4.5.3. This field is reserved unless the Enable Header Generation field has a value of 5, 6, or 7. If this field is set, then Write Source 2 must have a value of 1, Read Source 2 must have a value of 1, and the following fields are reserved: Load Dictionary, Enable Zero-Compress-8, Enable Indexing, Compress Bit Order, Generate All Literals, End Processing, and Early Abort Threshold.
14:12	**Enable Header Generation** **0**: Header Generation is disabled. **1:** 1-Pass Header Generation: Do not generate a Deflate Header **2:** 1-Pass Header Generation: Generate a Deflate Header **3**: 1-Pass Header Generation: Generate a bFinal Deflate Header **4**: Reserved **5**: 2-Pass Header Generation: Do not generate a Deflate Header **6**: 2-Pass Header Generation: Generate a Deflate Header **7**: 2-Pass Header Generation: Generate a bFinal Deflate Header 1-Pass header generation **cannot** be used if: • The amount of data being compressed is more than 4kB, or • Load Dictionary is non-zero If 2-pass header generation is selected, then the Write AECS Huffman Tables flag in Compression 2 Flags must be set.
11:10	**Load Dictionary** **0:** Do not load a dictionary. **1:** Load a 2kB dictionary with 2 pointers/hash table entry. **2:** Load a 4kB dictionary with 2 pointers/hash table entry. **3:** Load a 4kB dictionary with 4 pointers/hash table entry. This field is reserved if Enable Header Generation has the values 1, 2, or 3, if Header Gen Stats Input is set, or if Enable Indexing is non-zero.
9	**Reserved:** Must be 0.

Compression Flags	
Bits	**Description**
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 it is the last descriptor in a job. This field is reserved if bit 9 (Enable Zero-Compress-8) is set, if Header Gen Stats Input is set, or if Load Dictionary is non-zero.
5	**Compress Bit Order** Specifies the bit order of the compression output. **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.) If this bit is set, then the output will consist of an even number of bytes. This field is reserved if Header Gen Stats Input is set.
4	**Generate All Literals** **0:** Generate literals and matches. **1:** Generate only literals. This results in only doing Huffman Compression. This field is reserved if Header Gen Stats Input is set.
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. This field is reserved if Header Gen Stats Input is set.
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. If it would overflow the output buffer, the operation returns an error. 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. This flag is reserved if Enable Header Generation is non-zero.

6.3.5.2 Compression 2 Flags

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

Table 6-13 Compression 2 Flags

Compression 2 Flags	
Bits	**Description**
31:7	**Reserved:** Must be 0.
6:5	**Early Abort Size** Check for Early Abort after the specified number of data bytes have been compressed: **0:** 512 **1:** 1024 **2:** 2048 **3:** 4096 This field is reserved unless Early Abort Threshold has a non-zero value.
4:2	**Early Abort Threshold** **0:** Disable early aborts **1:** Set the Early Abort threshold at 1/8. **2:** Set the Early Abort threshold at 2/8. **3:** Set the Early Abort threshold at 3/8. **4:** Set the Early Abort threshold at 4/8. **5:** Set the Early Abort threshold at 5/8. **6:** Set the Early Abort threshold at 6/8. **7:** Set the Early Abort threshold at 7/8. This field is reserved if Header Gen Stats Input is set or if Generate All Literals is set.
1	**Write AECS Huffman Tables** **0:** The size of the AECS written is the smaller of the specified size or 64 bytes. **1:** The size of the AECS written is the smaller of the specified size or 1568 bytes. • This flag is reserved unless Enable Header Generation is non-zero. • This flag is required if Enable Header Generation has the values 5, 6, or 7.
0	**Make Complete Tables** **0:** Do not modify input statistics. **1:** Modify input statistics as described in Section 3.4.5. This field is reserved unless Enable Header Gen has a non-zero value.

6.3.6 CRC64 Descriptor (0x44)

The CRC64 operation computes a programmable arbitrary CRC of up to 64 bits in size.

CRC-64 Descriptor

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
Source 2 Address								40
Source 2 Transfer Size				Reserved				48
CRC Polynomial								56

Figure 6-7 CRC-64 Descriptor

The CRC Flags are described in Table 6-14.

CRC-64 Completion Record

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

Figure 6-8 CRC-64 Completion Record.

The CRC64 field contains the CRC result, in the most-significant part of the field, as defined by the CRC Bit Order flag. See Section 3.10.3 for details.

6.3.6.1 CRC Flags

Note that the flag bits 8:2,0 are identical to the Decompression Flags (Section 6.3.2.1).

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

Table 6-14 CRC Flags

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.
13:10	**Reserved:** Must be 0.

CRC Flags	
Bits	**Description**
9:6	**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. A value larger than 7 is only allowed if Decompress Bit Order is set. In this case, the value gives the number of bits to drop in the last 16-bit word.
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, then 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.
1	**Reserved: Must be 0.**
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, CRC Flag bits [9:2] are reserved.

6.3.6.2 CRC Polynomial

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

This field defines the polynomial for the CRC, as described in Section 3.10.3. 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.

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, Encrypt, Decrypt, and filter opcodes, and one for the compression opcode.

6.4.1 AECS Format for Encrypt, Decrypt, Decompress, and Filter

When the amount of AECS data read is less than the full amount, the unread fields receive default values. See Section 6.4.1.2 for details.

The format of the AECS for decompress and filter is shown in Table 6-15.

Table 6-15 AECS Format for Decompress and Filter

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 Bytes		0x14	
Crypto Accumulator Sizes		Crypto Flags	Crypto Algorithm	0x18	Crypto
Reserved				0x1C	
Crypto Input Accumulator				0x20	
				0x34	
Crypto Output Accumulator				0x38	
				0x44	
AES Key Low				0x48	
				0x54	
AES Key High				0x58	
				0x64	
Reserved				0x68	
				0x74	
Counter/IV				0x78	
				0x84	
GCM H				0x88	
				0x94	
Hash				0x98	
				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	

Byte 3	Byte 2	Byte 1	Byte 0	Byte Offset	Category
Reserved		AECS Format	Drop Initial Bits	0x1DC	
Decompression Internal State				0x1E0	
				0x14E4	

Table 6-16 AECS Fields for Decompress and Filter

Field	Description
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 Bytes	The number of initial bytes in the Filter Input that should be dropped before starting the filter operation. See Section 3.7. If the operation is Encrypt or Decrypt, then Drop Initial Bytes must be 0.
Drop Initial Bits	The number of initial bits in the decompress or filter input that should be dropped before starting the decompress/filter operation. See Section 3.7. If the operation is CRC64, and decompression is not enabled, then Drop Initial Bits must be 0. If the operation is Encrypt or Decrypt, then Drop Initial Bits must be 0.
AECS Format	**AECS Format:** **0:** Format-1 **1:** Format-2 This field is the low-order 3 bits within the specified byte. See Section 3.2.1 for information on how to use this field.
Cryptographic Algorithm	**Cryptographic Algorithm** **0:** GCM **1**: AES-CFB **2:** XTS **Other:** Reserved
Crypto Flags	**Bits:** **0:** Enable Crypto: 1=Enable Crypto **1:** Key_size_256: 0=key size 128 bits, 1=key size 256 bits **2:** Reserved **3:** Flush Crypto Input Accumulator: 1=Flush Crypto Input Accumulator **7:4**: Reserved • These bits are all reserved except for the following operations: Decrypt, Encrypt, Decompress, CRC64, Scan, and Extract. • Enable Crypto is required for operations: Encrypt and Decrypt. • If Enable Crypto is set, then the Operation Flag “Read Source 2” must have a value 1. • For operations other than Decrypt, Encrypt, and Decompress, Flush Crypto Input Accumulator must be set.
Crypto Accumulator Size	**Bits 5:0:** Number of valid bytes in the Crypto Input Accumulator **Bits: 15:6:** Reserved This value must be less than or equal to 40. For operations other than Decrypt, Encrypt, and Decompress, this field must be 0.
Crypto Input Accumulator	Input data for the crypto unit that has not yet been processed

Field	Description
AES Key Low	AES 128-bit key, or low 128 bits of AES 256-bit key
AES Key High	High 128 bits of AES 256-bit key
Counter/IV	Value of the AES counter/IV. Initial value is read, final value is written.
GCM H	GCM “H” parameter. This is the encrypted value of all zeros.
Hash	For GCM, this is the value of the hash. The initial value is read, and the final value is written.
Output Accumulator Data	Output data that has not yet been written to the output stream.
Output Accumulator Bits Valid	Number of valid data bits in output accumulator. Bit 0 is treated as the AECS Format Valid bit; see Section 3.2.1. For the purposes of the number of valid bits, it is treated as if it were 0. Since the Output Accumulator contains whole bytes, the only valid values are: 0, 8, 16, 24, 32, 40, 48, and 56.
Bit Offset for Indexing	Total number of consumed bits on input.
Input Accumulator Data	Input Deflate data that has not yet been processed.
Size QWn	The number of bytes valid in the corresponding Quadword in the Input Accumulator. Valid values are 0-64.
Decompress Internal State	Contains internal state of the Analytics hardware required to link together multiple segments that belong to one logical file.

6.4.1.1 Decompress Internal State

The Decompress/Analytics Internal State is shown in Table 6-17.

Table 6-17 Decompress/Analytics Internal State

Format-1 Field	Format-2 Field	Byte Offset
EOB CAM Entry		0x1D8
AECS Format/Drop Initial Bits		0x1DC
ALU First Table Index	Reserved	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
Decompression Performance Hints		0x324
LL Mapping Table	Reserved	0x328-0x33C
LL Mapping Table	LL Mapping CAM Part 1	0x340-0x41C
LL Mapping Table	Reserved	0x420-0x430
Decompress State		0x434
Stored Block Bytes Remaining		0x438
Reserved		0x43C
Reserved	LL Mapping CAM Part 2	0x440-0x4CC
Reserved		0x4D0-0x4E0
History Buffer Write Pointer		0x4E4

Format-1 Field	Format-2 Field	Byte Offset
History Buffer		0x4E8-0x14E4

The format of the LL and Distance 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 6-18. The value in each field is stored towards the least-significant end.

Table 6-18 ALU Field Definitions

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

The basic idea behind each ALU is that (in the non-bit-reversed space) all of the codes of a given length are sequential values. So, these can be represented as a first code and a number of codes.

In Format 1, 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. So, the ALU must 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.

In Format 2, there is a mapping CAM. The index of the CAM entry is implicitly the value of the literal or length. For example, CAM entry 0 defines the code for literal 00. For each CAM entry, the low order 4 bits contains the code length. The higher order bits contain an index indicating how many codes of that length occurred previously. In the above example, the three CAM entries would contain values 0x04, 0x14, 0x24. The location of these codes in the CAM depends on what three literal values were being represented.

Both the mapping table and CAM are only used for LL codes in the range of 0…264 (i.e., each one contains 265 entries). These correspond to the LL codes that do not contain extra bits. The LL codes that are followed by extra bits (i.e., LL[265]…LL[285]) are handled via the LL CAM.

The actual fields are described in Table 6-19:

Table 6-19 ALU Field Descriptions

Field	Description
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.
Decompress Performance Hints	Decompress Performance Hints. See Section 6.4.1.2 for details.
LL Mapping Table (Format 1)	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.
LL Mapping CAM (Format 2)	Each entry contains key value as describe above.
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.
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. If the Decompress State is 0010 or 0110, then this field must be non-zero.
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.
History Buffer	History Buffer.

6.4.1.2 Decompression Performance Hints

The nibbles of this field provide “hints” to the decompressor to improve its performance. They are “hints” in the sense that if they are set incorrectly or not set at all, then the decompression will still produce the correct output, but it may take longer than if they are set correctly.

In AECS Format-1, there is only the first hint present (bits 3:0). In Format-2, there is a varying number of hints depending on the implementation. Setting a non-existent hint has no effect. That is, there is nothing wrong with settling a hint that isn't supported.

The hints are:

Bits 3:0	Total length (code length plus number of extra bits) of the shortest Length Code.
Bits 7:4	Total length of the shortest Literal Code
Bits 11:8	Total Length of the second Shortest Literal Code. This may be the same length as for the shortest Literal code if there are more than one literal codes with the shortest length.

6.4.1.3 Default AECS Values

Other than for the following fields, the default AECS values are 0.

Table 6-20 Default Field Values

Field	Value (Format 1)	Value (Format 2)
Output Bits Valid	0	1
AECS Format	0	1
Decompress State	Expecting Start of Deflate Header	
Internal State	ALUs and CAMs configured for Deflate Fixed Codes	

6.4.2 AECS Format for Compress

The format of the AECS for compress is shown in Table 6-21.

Table 6-21 AECS Format for Compress

Byte 3	Byte 2	Byte 1	Byte 0	Byte Offset	Category
CRC				0x0	Checksums
Reserved		XOR Checksum		0x4	
Reserved				0x8	
Reserved				0x18	
Reserved	Last	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	
Reserved				0x618	
Reserved				0x61C	
Start Dictionary Data				0x64620	Dictionary
--				Varies	

Table 6-22 AECS Fields for Compress

Field	Description
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.
Last Descriptor Bit	This indicates the last descriptor for compression indexing. See Section 3.4.7 for details.
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. This value must be no greater than 2048.
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**.**
Start Dictionary Data	Start of variable sized region containing dictionary data. See Section 3.4.4 for details.

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 the case of an error, the entire operation must be re-executed. The two exceptions to this are the Decompress and Translation Fetch 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 Translation Fetch, 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 must 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.
- 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.
- **Inter-Domain Support:** Intel IAA does not support Inter-Domain features.

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)

Table 7-1 General Capabilities Register (GENCAP) Description

GENCAP			
Base: BARO	**Offset: 0x10**		**Size: 8 bytes (64 bits)**
Bit	**Attr**	**Size**	**Description**
63:42	RO	22 bits	Unused
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.

7.2.2 Intel IAA Capabilities Register (IAACAP)

Table 7-2 Intel® IAA Capabilities Register (IAACAP) Description

IAACAP			
Base: BARO	**Offset: 0x180**		**Size: 8 bytes (64 bits)**
Bit	**Attr**	**Size**	**Description**
63:12	RO	52 bits	Unused.
11	RO	1 bit	**Crypto XTS Support** **0:** Encryption and Decryption are not supported with the XTS algorithm. **1:** Encryption and Decryption are supported with the XTS algorithm. Typically, the OPCAP bit for ENCRYPT and DECRYPT would be cleared if no crypto algorithm was supported.
10	RO	1 bit	**Crypto CFB Support** **0:** Encryption and Decryption are not supported with the CFB algorithm **1:** Encryption and Decryption are supported with the CFB algorithm. Typically, the OPCAP bit for ENCRYPT and DECRYPT would be cleared if no crypto algorithm was supported.
9	RO	1 bit	**Crypto GCM Support** **0:** Encryption and Decryption are not supported with the GCM algorithm **1:** Encryption and Decryption are supported with the GCM algorithm. Typically, the OPCAP bit for ENCRYPT and DECRYPT would be cleared if no crypto algorithm was supported.
8	RO	1 bit	**Header Generation Support** **0:** Compression Header Generation is not supported. **1:** Compression Header Generation (see Section 3.4.5) is supported. This value should be ignored if the COMPRESS opcode is not supported.
7	RO	1 bit	**Dictionary Compression Support** **0:** Compression with a dictionary is not supported. **1:** Compression with a dictionary (see Section 3.4.4) is supported. This value should be ignored if the COMPRESS opcode is not supported.

IAACAP			
Base: BARO	**Offset: 0x180**		**Size: 8 bytes (64 bits)**
Bit	**Attr**	**Size**	**Description**
6	RO	1 bit	Unused.
5	RO	1 bit	**Compression Early Abort Support** **0:** Compression Early Abort is not supported. **1:** Compression Early Abort (see Section 3.4.6) is supported. This value should be ignored if the COMPRESS opcode is not supported.
4	RO	1 bit	**Load Partial AECS Support** **0:** If the Decompress Flag bit 13 (Load Partial) is set, then the descriptor will return an error. **1:** Decompress Flag bit 13 (Load Partial) can be used.
3	RO	1 bit	**Force Array Output Modification Support** **0:** If the Filter Flag bit 27 (Force Array Output Modification) is set, then the descriptor will return an error. **1:** Filter Flag bit 27 (Force Array Output Modification) can be used.
2	RO	1 bit	**Chaining to CRC64** **0:** Operation CRC64 cannot be preceded by any other operation. **1:** This operation can be preceded by some combination of Decrypt and Decompress. See Section 3.1.
1	RO	1 bit	**Drop Initial Bits Support** **0:** If the Drop Initial Bits field in the AECS is non-zero, the descriptor will return an error. **1:** A non-zero value of the Drop Initial Bits field is allowed. See Section 6.4.1.
0	RO	1 bit	**Generation 2 Minimum Capabilities** **0:** Minimum Generation 2 features are not present, in particular: • The decompress internal state in the AECS is Format 1. In particular, the Literal/Length Mapping is represented by a table. • Completion Record checksum fields are written as 0 for CRC64. • CRC Select is reserved for the CRC64 opcode. • If the operation is other than No-op or Drain, at least one of Source 1 Transfer Size and Maximum Destination Size must be non-zero. • Ignore End Bits Extension is not supported. • Last Descriptor bit in the compression AECS is not present. **1:** Minimum Generation 2 features are present, in particular: • The decompress internal state in the AECS is Format 2. In particular, the Literal/Length Mapping is represented by a CAM. (See Section 6.4.1.1) • Completion Record checksum fields are written correctly for CRC64. • CRC Select can be used with the CRC64 opcode. • If the operation is other than No-op or Drain, at least one of Source 1 Transfer Size, Source 2 Transfer Size, and Maximum Destination Size must be non-zero. • Ignore End Bits Extension is supported. • Last Descriptor bit in the compression AECS is present. (See Section 3.4.7)

7.3 PCI Express (PCIe) Configuration Register Differences

7.3.1 Device ID (DID)

The Device ID (DID) Identifies the particular device.

DEVICE ID (DID)				
Base: Rootbus	**CFG Offset:** 0x2			**Size:** 2 bytes (16 bits)
Default Value: 0x0CFE				
Bits	**Attr**	**Size**	**Default Value**	**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) is the maximum number of outstanding page requests.

PRSREQCAP				
Base: Rootbus	**CFG Offset:** 0x248			**Size: 4** bytes (32 bits)
Default Value: Implementation Defined				
Bits	**Attr**	**Size**	**Default Value**	**Description**
31:0	RO	32	Implementation Defined	**Capacity (CAP)** How many page requests the function can issue.