**DirectX Video Acceleration Specification  
for AV1 Video Coding**

September 2019

Matthew Wozniak, Abo-Talib Mahfoodh

© 2019 Microsoft Corporation.  All rights reserved.  This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it. This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

*Abstract – This document contains a specification for support of AV1 video decoding within the Microsoft Windows DirectX Video Acceleration (DXVA) API/DDI context. This specification includes support of the AV1 Main Profile (in 8-bit and 10-bit). The document describes high-level design concepts and specific AV1 extensions to DXVA interfaces and data structures of AV1 video decoding. This document specifies only off-host Variable Length Decoding (VLD) profiles for AV1 video decoding.*

Contents

[1. Introduction - 1 -](#_Toc19267364)

[1.1 Referenced Documents and Software - 1 -](#_Toc19267365)

[1.2 General Design Considerations - 2 -](#_Toc19267366)

[1.3 Support Only for Off-Host VLD Operation - 2 -](#_Toc19267367)

[1.4 Picture Data - 2 -](#_Toc19267368)

[1.5 Buffer Types - 2 -](#_Toc19267369)

[1.6 DXVA Decoding Operations - 2 -](#_Toc19267370)

[1.7 Status Reporting - 4 -](#_Toc19267371)

[1.8 Accelerator Internal Operations and Information Storage - 4 -](#_Toc19267372)

[1.9 Configuration Parameters - 4 -](#_Toc19267373)

[1.9.1 Syntax - 4 -](#_Toc19267374)

[1.9.2 Semantics - 5 -](#_Toc19267375)

[1.9.3 Accelerator Decoder Specific Support - 6 -](#_Toc19267376)

[1.10 Film Grain - 8 -](#_Toc19267377)

[2. DXVA\_PicEntry\_AV1\_Short Data Structure - 8 -](#_Toc19267378)

[2.1 Syntax - 9 -](#_Toc19267379)

[2.2 Semantics - 9 -](#_Toc19267380)

[3. DXVA\_PicEntry\_AV1 Data Structure - 9 -](#_Toc19267381)

[3.1 Syntax - 9 -](#_Toc19267382)

[3.2 Semantics - 10 -](#_Toc19267383)

[4. AV1 Picture Parameters Data Structure - 11 -](#_Toc19267384)

[4.1 Syntax - 12 -](#_Toc19267385)

[4.2 Semantics - 15 -](#_Toc19267386)

[4.2.1 General Picture Parameters - 15 -](#_Toc19267387)

[4.2.2 Tiles - 16 -](#_Toc19267388)

[4.2.3 Format and Picture Information Flags - 16 -](#_Toc19267389)

[4.2.4 Coding Tools flags - 17 -](#_Toc19267390)

[4.2.5 Reference Pictures - 21 -](#_Toc19267391)

[4.2.6 Loop Filter Parameters - 22 -](#_Toc19267392)

[4.2.7 CDEF Parameters - 23 -](#_Toc19267393)

[4.2.8 Quantization Parameters - 24 -](#_Toc19267394)

[4.2.9 Segmentation Parameters - 24 -](#_Toc19267395)

[4.2.10 Film Grain Parameters - 25 -](#_Toc19267396)

[4.2.11 Miscellaneous Parameters - 28 -](#_Toc19267397)

[5. Tile Control Data Structure - 28 -](#_Toc19267398)

[5.1 Syntax - 28 -](#_Toc19267399)

[5.2 Semantics - 29 -](#_Toc19267400)

[6. Status Report Data Structure - 29 -](#_Toc19267401)

[6.1 Syntax - 30 -](#_Toc19267402)

[6.2 Semantics - 30 -](#_Toc19267403)

[7. Restricted-Mode Profiles - 32 -](#_Toc19267404)

[7.1 DXVA\_ModeAV1\_VLD\_Profile0 Profile - 32 -](#_Toc19267405)

[7.2 DXVA\_ModeAV1\_VLD\_Profile1 Profile - 32 -](#_Toc19267406)

[7.3 DXVA\_ModeAV1\_VLD\_Profile2 Profile - 33 -](#_Toc19267407)

[7.4 DXVA\_ModeAV1\_VLD\_12bit\_Profile2 Profile - 33 -](#_Toc19267408)

[7.5 DXVA\_ModeAV1\_VLD\_12bit\_Profile2\_420 Profile - 34 -](#_Toc19267409)

[8. For More Information - 34 -](#_Toc19267410)

# 1. Introduction

This specification defines extensions to DirectX® Video Acceleration (DXVA) to support decoding of AV1 video. See, e.g., [https://en.wikipedia.org/wiki/](https://en.wikipedia.org/wiki/AV1)AV1 and <https://aomediacodec.github.io/av1-spec/av1-spec.pdf>.

This specification assumes the reader is familiar with the AV1 specification, and with the basic design of DXVA. This specification of DXVA usage for AV1 is designed to be maximally consistent with prior DXVA schemes for other video formats.

DXVA consists of a DDI for display drivers and an API for software decoders. Version 1.0 of DXVA is supported in Windows 2000 or later versions. Version 2.0 is available starting in Windows Vista. Considering the passage of time and the increasing prevalence of DXVA 2.0 support, this document specifies only the DXVA 2.0 operation for AV1 video decoding. We do not plan to specify AV1 video decoding in the DXVA 1.0 context.

In DXVA, some decoding operations are implemented by the graphics hardware driver and GPU. This set of functionalities is termed the *accelerator*. Other decoding operations, such as frame surface allocation, retirement, reuse, and release, are implemented by user-mode application software, called the *host* decoder or *software decoder*. Processing performed by the accelerator is sometimes referred to as *off-host* processing. Typically, the accelerator uses the GPU to speed up some operations. When the *accelerator* performs a decoding operation, the *host* decoder sends buffers of parameters and data to the accelerator that contain the information that is needed to perform the operations.

Except where stated otherwise in this specification, DXVA operations in the accelerator shall be *stateless*; the accelerator design shall not contain assumptions about the sequences of decoding operation or internal-memory state dependencies. This is necessary to enable good "trick play" and loss/error resilience functionality.

**Note –** In this document, the term *shall* describes behavior that is required by the specification. The term *should* describes behavior that is encouraged but not required. The term *note* refers to observations about implications of the specification.

Questions or comments about this specification may be sent to [askdxva@microsoft.com](mailto:askdxva@microsoft.com).

## 1.1 ***Referenced Documents and Software***

There is a document published by the Alliance For Open Media that describes the AV1 video format and decoding process completely:

<https://aomediacodec.github.io/av1-spec/av1-spec.pdf>

Associated open-source reference software for AV1 is available at the following link:

[*https://aomedia.googlesource.com/*](https://aomedia.googlesource.com/)

This software is currently considered the primary definitive reference for these formats. In this specification, the phrase "reference software" refers to this publicly available software.

## 1.2 General Design Considerations

Section 1 of this specification provides an overview of the DXVA design for AV1 video decoding. It is intended as background information, and may be helpful in understanding the sections that follow. In the case of conflicts, later sections of this document override this section. The initial design documented here is intended to be sufficient for decoding AV1 bitstreams in Main Profile. AV1 does not include support for interlaced-scan field-based coding or field-based display – it is designed for progressive-scan coding and display only. Hence in this specification a picture is considered synonymous with a frame.

## 1.3 Support Only for Off-Host VLD Operation

Over time, the level of industry interest in supporting modes of DXVA operation other than off-host Variable Length Decoding (VLD) operation (e.g., as in the DXVA\_ModeH264\_MoComp\_NoFGT and DXVA\_ModeH264\_IDCT\_NoFGT profiles of DXVA operation for H.264/AVC video decoding, and the DXVA\_ModeWMV9\_PostProc and DXVA\_ModeVC1\_IDCT profiles of DXVA operation for WMV9/VC-1 video decoding) appears to have waned. We therefore do not plan to specify such modes of DXVA operation for AV1 video decoding; only off-host VLD mode of DXVA operation is specified for AV1 video decoding.

## 1.4 Picture Data

Picture data must be conveyed for AV1 frame in order to decode each frame independently without serial dependencies or with minimized serial dependencies. For further details, see section 3 (AV1 Picture Parameters Data Structure) of this specification.

## 1.5 Buffer Types

The host software decoder will send the following DXVA buffers to the accelerator in off-host VLD operation:

* One picture parameters buffer (**DXVA\_PicEntry\_AV1**).
* One tile control (slice control) buffer (**DXVA\_Tile\_AV1**).
* One or more bitstream data buffers.

These buffer types are defined as in the prior DXVA specifications, but new data structures for the data carried within them have been defined herein for AV1 video decoding. The sequence of operations is described in section 1.6.

## 1.6 DXVA Decoding Operations

The basic sequence of operations for DXVA decoding consists of the following calls by the host software decoder. In DXVA 2.0, they are part of the **IDirectXVideoDecoder** interface.

1. **BeginFrame**. Signals the start of one or more decoding operations by the accelerator, which will cause the accelerator to write data into an uncompressed surface buffer.
2. **Execute**. The decoder calls **Execute** one time, sending one compressed data buffer to the accelerator and specifying the operations to perform on the buffer. The accelerator may return status information from the call. In DXVA 2.0, the command is specified in the **Function** member of the optional **DXVA2\_DecodeExtensionData** structure passed to **IDirectXVideoDecoder::Execute** by the **DXVA2\_DecodeExecuteParams** structure.
3. **EndFrame**. Signals that the host software decoder has sent all of the data needed for the corresponding **BeginFrame** call.

For AV1 video decoding, the data passed with the **Execute** method includes a destination index to indicate which uncompressed surface buffer is affected by the operation. The host software decoder can call **Execute** more than once between each **BeginFrame**/**EndFrame** pair. The host software decoder shall send the data for exactly one compressed frame between each **BeginFrame**/**EndFrame** pair.

When processing a frame of data, the accelerator will, in some cases, access uncompressed surfaces other than the surface being written to. For example, decoding a frame may require data from one or more previously-decoded frames for use as reference data for inter-picture motion-compensated prediction. If the host software decoder issues a command that requires writing to a buffer, and then issues a command that requires reading from the same buffer, it is the accelerator's responsibility to serialize these operations. In other words, the accelerator must complete a preceding write operation before starting a subsequent read operation on the same buffer.

The DXVA design for AV1 video decoding restricts the sequence of buffer types that can be sent to the accelerator. With compressed picture decoding in off-host parsing, i.e., with VLD profile operation, the host software decoder sends the following data buffers:

* One picture parameters data buffer.
* One tile control (slice control) data buffer.
* One bitstream data buffer.

The host software decoder does not send buffers for status reporting feedback. Rather, it reads such buffers when requesting status reporting feedback. Two values of *bDXVA\_Func* are defined, as follows:

|  |  |
| --- | --- |
| **Value** | **Description** |
| 1 | Compressed picture decoding with off-host parsing |
| 7 | Request for status report. |

*dwFunction* shall contain exactly one of the two values listed here. Function 7 (status reporting) is described in the next section.

Between a single pair of **BeginFrame** and **EndFrame** calls, the host software decoder can send one or more sets of buffers with *bDXVA\_Func* equal to 1 for off-host parsing.

The total quantity of data in any bitstream data buffer (and the amount of data reported by the host software decoder) shall be an integer multiple of 128 bytes. If the amount of source data is not an integer multiple of 128 bytes, the host shall append zero-valued bytes to the data so that this requirement is fulfilled. The accelerator shall ignore any such bytes that are present.

Whenever the host software decoder calls **Execute** to pass a set of compressed buffers to the accelerator, the private output data pointer shall be NULL, as stated in other DXVA 2.0 documentation: when the **NumCompBuffers** member of the **DXVA2\_DecodeExecuteParams** structure is greater than zero, *pPrivateOutputData* shall be NULL and *PrivateOutputDataSize* shall be zero. Alternatively, the **pExtensionData** member of the **DXVA2\_DecodeExecuteParams structure** can be NULL.

## 1.7 Status Reporting

After calling **EndFrame** for the uncompressed destination surfaces, the host software decoder may call **Execute** with *bDXVA\_Func* = 7 to get a status report. The host software decoder does not pass any compressed buffers to the accelerator in this call. Instead, the host decoder provides a private output data buffer into which the accelerator will write status information. The decoder provides the output data buffer as follows in DXVA 2.0: the host software decoder sets the *pPrivateOutputData* member of the DXVA2\_DecodeExecuteParams structure to point to the buffer. The *PrivateOutputDataSize* member specifies the maximum amount of data that the accelerator is allowed to write to the buffer. The value of *cbPrivateOutputData* or *PrivateOutputDataSize* shall be an integer multiple of sizeof(**DXVA\_Status\_AV1**).

When the accelerator receives the **Execute** call for status reporting, it should not stall operation to wait for any prior operations to complete. Instead, it should immediately provide the available status information for all operations that have completed since the previous request for a status report, up to the maximum amount requested. Immediately after the **Execute** call returns, the host software decoder can read the status report information from the buffer. The status report data structure is described in section 6.

## 1.8 Accelerator Internal Operations and Information Storage

The AV1 decoding process requires storing some additional information along with the array of decoded frames to be used as reference pictures for picture decoding. Rather than have the host decoder collect this information and explicitly update and provide it to the accelerator, the accelerator shall store this information as it decodes each picture, so that the information is available if the picture is later used as a reference picture.

## 1.9 Configuration Parameters

This section describes the configuration parameters for AV1 video decoding according to this specification.

### 1.9.1 Syntax

In DXVA 2.0, configuration uses the DXVA2\_ConfigPictureDecode structure. This syntax structure is documented in the DXVA 2.0 documentation, available at <http://msdn.microsoft.com/en-us/library/ms694823(VS.85).aspx>.

### 1.9.2 Semantics

The ordinary semantics of this data structure apply for AV1 video decoding according to this specification. Details of the usage in this context are provided below.

**guidConfigBitstreamEncryption**

Defines the encryption protocol type for bitstream data buffers. If no encryption is applied, the value is DXVA\_NoEncrypt.

**guidConfigMBcontrolEncryption**

Shall be DXVA\_NoEncrypt, as **ConfigBitstreamRaw** is equal to 1 always.

**guidConfigResidDiffEncryption**

Shall be DXVA\_NoEncrypt, as **ConfigBitstreamRaw** is equal to 1 always.

**ConfigBitstreamRaw**

Shall be 1, as only off-host VLD parsing profiles are supported by this specification with **DXVA\_Slice\_AV1\_Short** structure for AV1 video decoding.

**ConfigMBcontrolRasterOrder**

Shall be 0, as **ConfigBitstreamRaw** is equal to 1 always.

**ConfigResidDiffHost**

Shall be 0, as **ConfigBitstreamRaw** is equal to 1 always.

**ConfigSpatialResid8**

Shall be 0, as **ConfigResidDiffHost** is equal to 0 always.

**ConfigResid8Subtraction**

Shall be 0, as **ConfigSpatialResid8** is equal to 0 always.

**ConfigSpatialHost8or9Clipping**

Shall be 0, as **ConfigResidDiffHost** is equal to 0 always.

**ConfigSpatialResidInterleaved**

Shall be 0, as **ConfigResidDiffHost** is equal to 0 always.

**ConfigIntraResidUnsigned**

Shall be 0, as **ConfigResidDiffHost** is equal to 0 always.

**ConfigResidDiffAccelerator**

Shall be 0, as **ConfigBitstreamRaw** is equal to 1 always.

**ConfigHostInverseScan**

Shall be 0, as **ConfigResidDiffAccelerator** is equal to 0 always.

**ConfigSpecificIDCT**

Shall be 0, as **ConfigResidDiffAccelerator** is equal to 0 always.

**Config4GroupedCoefs**

Shall be 0, as **ConfigResidDiffAccelerator** is equal to 0 always

**ConfigDecoderSpecific**

See below.

### 1.9.3 Accelerator Decoder Specific Support

The ConfigDecoderSpecific member of the **DXVA2\_ConfigPictureDecode** structure contains information about some decoder accelerator specific support. ConfigDecoderSpecific has the type unsigned short, where the least-significant bit is considered bit 0 and the most significant bit is bit 15.

For purposes specified herein, a "format change" is defined as the detection by the host decoder that the number of surfaces to be used has increased or that the maximum decoding resolution (picture max\_width or max\_height) has changed or that the accelerator capability requirements have changed, such as enabling or disabling downsampling of the output. AV1 decoders are required to support changing decoding resolution (picture width and height) within picture max\_width and max\_height. Bit 15 indicates that stream global parameters may change without decoder re-creation.

The semantics of bit 15 are as follows:

* 0b: in the event of a format change, some accelerators indicating this value may not be capable of continuing operation. The host decoder should therefore create a new video decoder device and destroy the old video decoder device when a format change occurs.
* 1b: in the event of a format change, the accelerator is indicated to be capable of continuing operation. The host decoder should not create a new video decoder device and proceed using the existing video decoder device instance.

When bit 15 of ConfigDecoderSpecific is set equal to 1 by the accelerator through the API GetVideoDecoderConfig(), the video decoder device can be reused after a format change and the host decoder should not create a new video decoder device in the event of a format change (thereby reducing latency relative to that experienced by recreating the decoder device).

Note – Older accelerators use the value 0 for bit 15 of ConfigDecoderSpecific, as the use of the value 1 was not defined prior to late 2015.

For purposes specified herein, an "array of textures" is defined as having ArraySize equal to 1 in the data structure of D3D11\_TEXTURE2D\_DESC, and a "texture array" is defined as having ArraySize equal to the number of needed surfaces in the data structure of D3D11\_TEXTURE2D\_DESC.

The semantics of bit 14 are as follows:

* 0b: accelerator may only support a texture array, or supports both an array of textures or a texture array for uncompressed surfaces but the use of a texture array may have better performance than an array of textures. In this case, the host decoder should create a texture array for uncompressed surfaces to ensure proper operation.
* 1b: accelerator supports both array of textures and texture array for uncompressed surfaces but an array of textures may have better performance than a texture array. In this case, the host decoder should create an array of textures for uncompressed surfaces.

The performance of the use of a "texture array" versus an "array of textures" may be different for different accelerators. Bit 14 of ConfigDecoderSpecific indicates the recommended configuration for the uncompressed surfaces used for decoding. The recommended value for bit 14 of ConfigDecoderSpecific is set by the accelerator through the API GetVideoDecoderConfig().

Note – Older accelerators use the value 0 for bit 14 of ConfigDecoderSpecific, as the use of the value 1 was not defined prior to late 2015.

The semantics of bit 13 are as follows:

* 0b: the accelerator does not support per-tile references. The contents of DXVA\_Tile\_AV1::anchor\_frame will never have a valid frame reference. Making “large scale tile decoding” support as defined in the specification, not possible.
* 1b: accelerator supports per-tile references. DXVA\_Tile\_AV1::anchor\_frame can contain valid references. And thus the accelerator supports “large scale tile decoding” as defined by the specification.

Note – Older accelerators use the value 0 for bit 13.

For purposes specified herein, a "format change on non-key frame" is defined as the detection by the host decoder that the decoding resolution (picture width or height) has changed on a non-key frame. The decoding process for frames with new resolution may refer to previous frames with a different resolution.

The semantics of bit 12 shall be referred to by the host decoder only if bit 15 is also set, and are as follows:

* 0b: in the event of a format change on a non-key frame, some accelerators indicating this value may not be capable of continuing operation. The host decoder should therefore drop frames till the next key frame.
* 1b: in the event of a format change on non-key frame, the accelerator is indicated to be capable of continuing operation.

The host decoder should not create a new video decoder device and proceed using the existing video decoder device instance. The host decoder will create new surfaces or reuse previously allocated surfaces with the new resolution and will hold reference to the older surfaces until needed.

An “array of textures” or “texture array” could be used for the uncompressed surfaces, which will be indicated by bit 14.

Note – Older accelerators use the value 0 for bit 12 of ConfigDecoderSpecific, as the use of the value 1 was not defined prior to 2016

## 1.10 Film Grain

Accelerators are expected to support frame post-process operations such as AV1’s film grain support. Frame post-process operations which are not used for reference require that the accelerator support separate Output Picture Buffers (OPB) and Display Picture Buffers (DPB).

On D3D11 this is achieved similarly to DXVA down sampling implemented in ID3D11VideoContext1::DecoderEnableDownsampling. When DXVA\_PicParams\_AV1::coding::film\_grain is set then the bitstream may use film grain features, and the decoder should allocate a DPB buffer which mirrors the OPB buffer provided by the host decoder. After **EndFrame** is called against a frame where DXVA\_PicParams\_AV1::film\_grain::apply\_grain is set, the decoder shall write the video frame without film grain applied to its internal buffer at DXVA\_PicParams\_AV1::CurrPicTextureIndex and also to the OPB buffer provided by the host at the same location.

On D3D12 the host decoder will always allocate and provide both an OPB and DBP buffer while film grain needs to be applied. This is done via D3D12’s video decode conversion support: D3D12\_VIDEO\_DECODE\_OUTPUT\_STREAM\_ARGUMENTS::ConversionArguments. After **EndFrame** is called against a frame where DXVA\_PicParams\_AV1::film\_grain::apply\_grain is set, the decoder shall write the video frame without film grain applied to the DPB buffer (D3D12\_VIDEO\_DECODE\_OUTPUT\_STREAM\_ARGUMENTS::ConversionArguments::pReferenceTexture2D) at DXVA\_PicParams\_AV1::CurrPicTextureIndex and also to the OPB buffer (D3D12\_VIDEO\_DECODE\_OUTPUT\_STREAM\_ARGUMENTS::pOutputTexture2D) at the same location.

# 3. DXVA\_PicEntry\_AV1 Data Structure

The **DXVA\_PicEntry\_AV1** structure specifies a reference to an uncompressed surface with additional metadata necessary for referencing that surface. It is used in other data structures described in this document. The form of this data structure is shown below.

## 3.1 Syntax

typedef struct \_DXVA\_PicEntry\_AV1 {

UINT width;

UINT height;

// Global motion parameters

INT wmmat[6];

union {

struct {

UCHAR wminvalid : 1;

UCHAR wmtype : 2;

UCHAR Reserved : 5;

};

UCHAR wGlobalMotionFlags;

};

UCHAR Index;

USHORT Reserved16Bits;

} DXVA\_PicEntry\_AV1, \*LPDXVA\_PicEntry\_AV1;

## 3.2 Semantics

**Index**

An index that identifies an index into **ref\_frame\_map\_texture\_index**[] to identify the frame used as a reference for this picture.

When **Index** does not contain an index to a valid reference, the value shall be set to 255, to indicate that the index is invalid.

**width**, **height**

Specify the coded width and height of the referred frame. These correspond to the syntax elements of: frame\_width\_minus\_1 and frame\_height\_minus\_1. If these values are derived for the frame (for example via the frame\_size\_override\_flag), the host decoder will derive the appropriate values and store the result here. Each AV1 frame may be coded at a different resolution than the previous frame(s).

**wmtype**

Specifies the global motion type of this referred frame. This matches the reference software’s TransformationType enum:

typedef enum {

IDENTITY = 0,

TRANSLATION = 1,

ROTZOOM = 2,

AFFINE = 3,

TRANS\_TYPES,

} TransformationType;

This attribute is informational, to help validate the contents of wmmat.

**wminvalid**

Specifies if the given global motion parameters are valid. If 0 then the parameters should be assumed to be invalid and not read.

**wmmat**

Specifies the global motion affine matrix as specified in the specification as: gm\_params. This is an affine transformation matrix which applies globally to motion vectors referring to this frame. (See section 7.10.2.1 of the specification).

# 4. AV1 Picture Parameters Data Structure

The **DXVA\_PicParams\_AV1** structure provides the picture-level parameters of a compressed picture for AV1 video decoding. This structure is used for AV1 when *bDXVA\_Func* is 1 and the buffer type is **DXVA2\_PictureParametersBufferType** (in DXVA 2.0).

## 4.1 Syntax

/\* AV1 picture parameters structure \*/

typedef struct \_DXVA\_PicParams\_AV1 {

UINT width;

UINT height;

UINT max\_width;

UINT max\_height;

UCHAR CurrPicTextureIndex;

UCHAR superres\_denom;

UCHAR bitdepth;

UCHAR seq\_profile;

// Tiles:

struct {

UCHAR cols;

UCHAR rows;

USHORT context\_update\_id;

USHORT widths[64];

USHORT heights[64];

} tiles;

// Coding Tools

union {

struct {

UINT use\_128x128\_superblock : 1;

UINT intra\_edge\_filter : 1;

UINT interintra\_compound : 1;

UINT masked\_compound : 1;

UINT warped\_motion : 1;

UINT dual\_filter : 1;

UINT jnt\_comp : 1;

UINT screen\_content\_tools : 1;

UINT integer\_mv : 1;

UINT cdef : 1;

UINT restoration : 1;

UINT film\_grain : 1;

UINT intrabc : 1;

UINT high\_precision\_mv : 1;

UINT switchable\_motion\_mode : 1;

UINT filter\_intra : 1;

UINT disable\_frame\_end\_update\_cdf : 1;

UINT disable\_cdf\_update : 1;

UINT reference\_mode : 1;

UINT skip\_mode : 1;

UINT reduced\_tx\_set : 1;

UINT superres : 1;

UINT tx\_mode : 2;

UINT use\_ref\_frame\_mvs : 1;

UINT enable\_ref\_frame\_mvs : 1;

UINT reference\_frame\_update : 1;

UINT Reserved : 5;

};

UINT32 CodingParamToolFlags;

} coding;

// Format & Picture Info flags

union {

struct {

UCHAR frame\_type : 2;

UCHAR show\_frame : 1;

UCHAR showable\_frame : 1;

UCHAR subsampling\_x : 1;

UCHAR subsampling\_y : 1;

UCHAR mono\_chrome : 1;

UCHAR Reserved : 1;

};

UCHAR FormatAndPictureInfoFlags;

} format;

// References

UCHAR primary\_ref\_frame;

UCHAR order\_hint;

UCHAR order\_hint\_bits;

DXVA\_PicEntry\_AV1 frame\_refs[7];

UCHAR RefFrameMapTextureIndex[8];

// Loop filter parameters

struct {

UCHAR filter\_level[2];

UCHAR filter\_level\_u;

UCHAR filter\_level\_v;

UCHAR sharpness\_level;

union {

struct {

UCHAR mode\_ref\_delta\_enabled : 1;

UCHAR mode\_ref\_delta\_update : 1;

UCHAR delta\_lf\_multi : 1;

UCHAR delta\_lf\_present : 1;

UCHAR Reserved : 4;

};

UCHAR ControlFlags;

} DUMMYUNIONNAME;

CHAR ref\_deltas[8];

CHAR mode\_deltas[2];

UCHAR delta\_lf\_res;

UCHAR frame\_restoration\_type[3];

USHORT log2\_restoration\_unit\_size[3];

UINT16 Reserved16Bits;

} loop\_filter;

// Quantization

struct {

union {

struct {

UCHAR delta\_q\_present : 1;

UCHAR delta\_q\_res : 2;

UCHAR Reserved : 5;

};

UCHAR ControlFlags;

} DUMMYUNIONNAME;

UCHAR base\_qindex;

CHAR y\_dc\_delta\_q;

CHAR u\_dc\_delta\_q;

CHAR v\_dc\_delta\_q;

CHAR u\_ac\_delta\_q;

CHAR v\_ac\_delta\_q;

// using\_qmatrix:

UCHAR qm\_y;

UCHAR qm\_u;

UCHAR qm\_v;

UINT16 Reserved16Bits;

} quantization;

// Cdef parameters

struct {

union {

struct {

UCHAR damping : 2;

UCHAR bits : 2;

UCHAR Reserved : 4;

};

UCHAR ControlFlags;

} DUMMYUNIONNAME;

union {

struct {

UCHAR primary : 6;

UCHAR secondary : 2;

};

UCHAR combined;

} y\_strengths[8];

union {

struct {

UCHAR primary : 6;

UCHAR secondary : 2;

};

UCHAR combined;

} uv\_strengths[8];

} cdef;

UCHAR interp\_filter;

// Segmentation

struct {

union {

struct {

UCHAR enabled : 1;

UCHAR update\_map : 1;

UCHAR update\_data : 1;

UCHAR temporal\_update : 1;

UCHAR Reserved : 4;

};

UCHAR ControlFlags;

} DUMMYUNIONNAME;

UCHAR Reserved24Bits[3];

union {

struct {

UCHAR alt\_q : 1;

UCHAR alt\_lf\_y\_v : 1;

UCHAR alt\_lf\_y\_h : 1;

UCHAR alt\_lf\_u : 1;

UCHAR alt\_lf\_v : 1;

UCHAR ref\_frame : 1;

UCHAR skip : 1;

UCHAR globalmv : 1;

};

UCHAR mask;

} feature\_mask[8];

SHORT feature\_data[8][8];

} segmentation;

struct {

union {

struct {

USHORT apply\_grain : 1;

USHORT scaling\_shift\_minus8 : 2;

USHORT chroma\_scaling\_from\_luma : 1;

USHORT ar\_coeff\_lag : 2;

USHORT ar\_coeff\_shift\_minus6 : 2;

USHORT grain\_scale\_shift : 2;

USHORT overlap\_flag : 1;

USHORT clip\_to\_restricted\_range : 1;

USHORT matrix\_coeff\_is\_identity : 1;

USHORT Reserved : 3;

};

USHORT ControlFlags;

} DUMMYUNIONNAME;

USHORT grain\_seed;

UCHAR scaling\_points\_y[14][2];

UCHAR num\_y\_points;

UCHAR scaling\_points\_cb[10][2];

UCHAR num\_cb\_points;

UCHAR scaling\_points\_cr[10][2];

UCHAR num\_cr\_points;

UCHAR ar\_coeffs\_y[24];

UCHAR ar\_coeffs\_cb[25];

UCHAR ar\_coeffs\_cr[25];

UCHAR cb\_mult;

UCHAR cb\_luma\_mult;

UCHAR cr\_mult;

UCHAR cr\_luma\_mult;

UCHAR Reserved8Bits;

SHORT cb\_offset;

SHORT cr\_offset;

} film\_grain;

UINT Reserved32Bits;

UINT StatusReportFeedbackNumber;

} DXVA\_PicParams\_AV1, \*LPDXVA\_PicParams\_AV1;

## 4.2 Semantics

### 4.2.1 General Picture Parameters

**CurrPicTextureIndex**

Specifies the destination frame buffer/surface index for the decoded picture.

**seq\_profile**

Indicates the profile of the AV1 bitstream. This corresponds to the syntax element: seq\_profile. The AV1 video coding format defines three profiles: profile 0, profile 1, profile 2. Profile 0 supports 4:2:0 [chroma sampling](http://en.wikipedia.org/wiki/Chroma_subsampling) with 8 or 10 bits per sample and monochrome formats. Profile 1 supports 4:4:4 chroma sampling with 8 or 10 bits per sample. And Profile 2 supports 4:2:0, 4:2:2, or 4:4:4 chroma sampling at 8, 10 or 12 bits per sample and monochrome formats.

**width**, **height**

Specify the coded width and height of the current frame. These correspond to the syntax elements of: frame\_width\_minus\_1 and frame\_height\_minus\_1. If these values are derived for the frame (for example via the frame\_size\_override\_flag), the host decoder will derive the appropriate values and store the result here. If superres is enabled these values represent the post-scaled frame resolution (referred to in the specification as UpscaledWidth).

**superres\_denom**

When superres is enabled specifies the denominator for computing the updated frame width. This corresponds to the SuperresDenom value from the specification. When superres is not enabled this value shall be 8.

**max\_width**, **max\_height**

Specify the maximum coded width and height of any frame in the sequence. These correspond to the syntax elements of: max\_frame\_height\_minus\_1, max\_frame\_width\_minus\_1 from the sequence header.

**bitdepth**

Indicate the bit depth of the luma and chroma decoded samples. The allowed values are restricted by the **profile** value.

### 4.2.2 Tiles

**cols, rows**

Specifies the number of tiles across (cols) and down (rows) a frame. These correspond to 2^TileColsLog2, and 2^TileRowsLog2 variables from the specification.

**widths, heights**

Specifies the widths and heights of each tile in units of superblocks. Only the first 2cols\_log2 widths entries are valid, and 2rows\_log2 heights entries are valid. These correspond to the tileWidthSb, and tileHeightSb variables from the specification.

**context\_update\_id**

Specifies which tile to use for CDF update. This corresponds to the syntax element named context\_update\_tile\_id from the specification.

### 4.2.3 Format and Picture Information Flags

**frame\_type**

Specifies the frame type of the current frame. It corresponds to the syntax element of the same name in the specification and affects the decoding process accordingly. The allowed values are 0, 1, 2 or 3, for the four types of AV1 frames: KEY\_FRAME, INTER\_FRAME, INTRA\_ONLY\_FRAME, and S\_FRAME.

**show\_frame**

Indicates whether the current frame is intended to be output and displayed after its decoding is completed. It corresponds to the syntax element of the same name in the specification and affects the decoding process accordingly. In DXVA, this has no direct effect, as the host controls the display of decoded frames separately by other function calls.

**showable\_frame**

TODO

**subsampling\_x, subsampling\_y**

Indicate the chroma sampling format. The allowed values of **subsampling\_x** and **subsampling\_y** are constrained by the **profile** value. The table below specifies the allowed values for subsampling\_x and subsampling\_y, and the associated chroma formats.

|  |  |  |
| --- | --- | --- |
| **subsampling\_x** | **subsampling\_y** | **Chroma format** |
| 1 | 1 | 4:2:0 |
| 1 | 0 | 4:2:2 |
| 0 | 0 | 4:4:4 |

**mono\_chrome**

Indicates weather the bitstream is monochrome, if 1 then only a luma plane is present. It corresponds to the syntax element of the same name from the Color Config Syntax of the AV1 specification. The allowed values are restricted by the **profile** value.

**Reserved**

Reserved bit fields to complete the packed structure. Shall be set to 0. The accelerator shall ignore the values in the reserved bit fields.

**FormatAndPictureInfoFlags**

Provides an alternative way to access the bit fields related to format and picture info.

### 4.2.4 Coding Tools flags

**use\_128x128\_superblock**

When equal to 1, indicates that superblocks contain 128x128 luma samples, otherwise superblocks contain 64x64 luma samples. It corresponds to the syntax element of the same name from the specification.

**intra\_edge\_filter**

Specifies weather the intra edge filtering process should be enabled. It corresponds to the syntax element named enable\_intra\_edge\_filter from the specification.

**interintra\_compound**

When equal to 1, indicates that the mode info for inter blocks may contain the syntax element interintra, otherwise the interintra syntax element will not be present. It corresponds to the syntax element named enable\_interintra\_compound from the specification.

**masked\_compound**

When equal to 1, indicates that the mode info for inter blocks may contain the syntax element compound\_type, otherwise the compound\_type syntax element will not be present. It corresponds to the syntax element named enable\_masked\_compound from the specification.

**warped\_motion**

When equal to 1, indicates that the syntax element motion\_mode may be present. It corresponds to the syntax element named allow\_warped\_motion from the specification.

**dual\_filter**

When equal to 1, indicates that the inter prediction filter type may be specified independently in the horizontal and vertical directions, otherwise only one filter type may be specified. It corresponds to the syntax element named enable\_dual\_filter from the specification.

**jnt\_comp**

When equal to 1, indicates that the distance weights process may be used for inter prediction. It corresponds to the syntax element named enable\_jnt\_comp from the specification.

**screen\_content\_tools**

When equal to 1, indicates that intra blocks may use palette encoding, otherwise palette encoding will not be used. It corresponds to the syntax element named allow\_screen\_content\_tools from the specification.

**integer\_mv**

When equal to 1, indicates that motion vectors will always be integers. It corresponds to the syntax element named force\_integer\_mv from the specification.

**cdef**

When equal to 1, indicates that cdef filtering may be enabled. It corresponds to the syntax element named enable\_cdef from the specification.

**restoration**

When equal to 1, indicates that loop restoration filtering may be enabled. It corresponds to the syntax element named enable\_restoration from the specification.

**film\_grain**

When equal to 1, indicates that film grain processing may be enabled. It corresponds to the syntax element named film\_grain\_params\_present from the specification.

**intrabc**

When equal to 1, indicates that intra block copy may be used. It corresponds to the syntax element named allow\_intrabc from the specification.

**high\_precision\_mv**

When equal to 1, indicates that motion vectors are specified to eighth pel precision, otherwise motion vectors are specified to quarter pel precision. It corresponds to the syntax element named allow\_high\_precision\_mv from the specification.

**switchable\_motion\_mode**

When equal to 1, indicates that only the SIMPLE motion mode may be used. It corresponds to the syntax element named is\_motion\_mode\_switchable from the specification.

**filter\_intra**

When equal to 1, indicates that the use\_filter\_intra syntax element may be present. It corresponds to the syntax element named enable\_filter\_intra from the specification.

**disable\_frame\_end\_update\_cdf**

Specifies weather the CDF arrays are saved in the symbol decoder exit process. It corresponds to the syntax element of the same name from the specification.

**disable\_cdf\_update**

Specifies weather the CDF update in the symbol decoding process should be disabled. It corresponds to the syntax element of the same name from the specification.

**reference\_mode**

Controls the mode used for reference frame prediction. It can take on two values which correspond with the reference mode from the AV1 decoding process:

|  |  |
| --- | --- |
| **reference\_mode** | **Name of reference\_mode** |
| 0 | SINGLE\_REFERENCE |
| 1 | REFERENCE\_MODE\_SELECT |

**skip\_mode**

If skip mode syntax can be specified in compound prediction. It corresponds to the skip\_mode\_present syntax element from the specification.

**reduced\_tx\_set**

Specifies if the frame uses a restricted set of transform types. It corresponds to the syntax element of the same name from the specification.

**superres**

Specifies if the frame uses super resolution. It corresponds to the syntax element named use\_superres from the specification.

**tx\_mode**

Specifies how the transform size is determined. It corresponds to the variable of the same name from the specification. It can take on values as specified in table 6.8.21 in the AV1 specification:

|  |  |
| --- | --- |
| **TxMode value** | **Name of TxMode** |
| 0 | ONLY\_4X4 |
| 1 | TX\_MODE\_LARGEST |
| 2 | TX\_MODE\_SELECT |

**use\_ref\_frame\_mvs**

Specifies weather motion vectors from the previous frame should be used for the current frame. It corresponds to the syntax element of the same name from the specification.

**enable\_ref\_frame\_mvs**

Specifies weather use\_ref\_frame\_mvs is disabled for the entire video sequence; when 0 use\_ref\_frame\_mvs will also always be 0. It corresponds to the syntax element of the same name from the specification (from the sequence header).

**reference\_frame\_update**

Indicates that the reference frame update process as specified by section 7.20 of the specification should be performed after decoding this frame. Otherwise section 7.21 should be performed.

**CodingParamToolFlags**

Provides an alternative way to access the bit fields related to coding tool flags.

**Reserved**

Reserved bit fields to complete the packed structure. Shall be set to 0. The accelerator shall ignore the values in the reserved bit fields.

### 4.2.5 Reference Pictures

**primary\_ref\_frame**

Specifies which reference frame contains the CDF values and other state that should be loaded at the start of the frame. It shall be 0x7 when there is no primary reference frame. It corresponds to the syntax element of the same name from the specification.

**order\_hint**

Corresponds to OrderHint from the specification. Specifies the output ordering of the current frame.

**order\_hint\_bits**

Corresponds to the syntax element named order\_hint\_bits\_minus\_1 in the specification. It is defined as: order\_hint\_bits\_minus\_1 + 1. This value will be 0 when enable\_order\_hint from the specification is 0. When 0 no tools based on OrderHint should be used and the value of order\_hint is undefined. When non-zero it affects the calculation of relative distance accordingly (see function get\_relative\_dist from the specification).

**RefFrameMapTextureIndex[]**

Contains a list of uncompressed frame buffer surfaces. Entries that will not be used for decoding the current picture, or any subsequent pictures, are indicated by setting this value to0xFF. If the valueis not 0xFF, the entry may be used as a reference surface for decoding the current picture or a subsequent picture in decoding order. All uncompressed surfaces that correspond to frames that may be used for reference in the decoding process of the current picture or any subsequent picture shall be present in the **RefFrameMapTextureIndex[]** array (regardless of whether these pictures are actually used in the decoding process of the current frame or not). No particular order is specified for the ordering of the entries in the **RefFrameMapTextureIndex[]** array.

**Note –** The accelerator must use the content of the **RefFrameMapTextureIndex[]** as provided by the accelerator rather than trying to derive this information from the bitstream (in order to ensure stateless operation for which the decoded frame buffer handling is to be performed under the control of the host rather than inferred from the bitstream by the accelerator).

**frame\_refs[]**

Indicates the reference surfaces to be used for inter prediction during current frame decoding. The reference surface indices in **frame\_refs[]** shall exist in **RefFrameMapTextureIndex[]** array.

**Note –** The AV1 decoder maintains a pool (**RefFrameMapTextureIndex[]**)of 8 reference pictures at all times. Each frame may pick up to 7 reference frames (**frame\_refs[]**)from the pool to use for inter prediction of the current frame. After the current frame finishes decoding, the host decoder can insert the current frame into any, all, or none of these 8 slots in the pool (**RefFrameMapTextureIndex[]**), evicting whatever frame was there before.

The accelerator shall use the content of the **frame\_refs[]** array as provided by the host decoder rather than trying to derive the information from the bitstream.

### 4.2.6 Loop Filter Parameters

**filter\_level[], filter\_level\_u, filter\_level\_v, sharpness\_level**

Correspond to the syntax elements of the same name in the specification and affect the decoding processes of the deblocking filter accordingly.

**mode\_ref\_delta\_enabled**

When equal to 1, indicates that the filter level depends on the mode and reference frame used to predict a block. It corresponds to the syntax element named loop\_filter\_delta\_enabled from the specification.

**mode\_ref\_delta\_update**

When equal to 1, indicates that additional syntax elements are present that specify which mode and reference frame deltas are to be updated. It corresponds to the syntax element named loop\_filter\_delta\_update from the specification.

**delta\_lf\_multi**

When equal to 1, indicates that separate loop filter deltas are sent for horizontal luma edges, vertical luma edges, U edges and V edges. It corresponds to the syntax element named delta\_lf\_multi from the specification.

**delta\_lf\_present**

When equal to 1, indicates that loop filter deltas are present. This is needed for the loop filter decoding process: read\_delta\_lf(). It corresponds to the syntax element named delta\_lf\_present from the specification.

**Reserved**

Reserved bit fields to complete the packed structure. Shall be set to 0. The accelerator shall ignore the values in the reserved bit fields.

**ControlFlags**

Provides an alternative way to access the bit fields related to loop filter flags.

**ref\_deltas[]**

Corresponds to the syntax element named loop\_filter\_ref\_deltas in the specification and affects the decoding processes of the loop filter accordingly.

**mode\_deltas[]**

Corresponds to the syntax element named loop\_filter\_mode\_deltas in the specification and affects the decoding process of the loop filter accordingly.

**delta\_lf\_res**

Specifies the left shift which should be applied to the decoded loop filter delta values. It corresponds to the syntax element of the same name from the specification.

**log2\_restoration\_unit\_size[]**

Specifies the log2 size of loop restoration units in units of samples in the current plane. Element 0 corresponds to the luma plane, element 1 to the U plane and element 2 to the V plane. It corresponds to the log2(LoopRestorationSize[plane]) variable from the specification.

**frame\_restoration\_type[]**

Specifies the loop restoration mode for this frame. It corresponds to the FrameRestorationType syntax element from the specification. It takes on the same values as specified in section 6.10.15 of the specification:

|  |  |
| --- | --- |
| FrameRestorationType | Name of FrameRestorationType |
| 0 | RESTORE\_NONE |
| 1 | RESTORE\_WIENER |
| 2 | RESTORE\_SGRPROJ |
| 3 | RESTORE\_SWITCHABLE |

### 4.2.7 CDEF Parameters

**damping**

Controls the amount of damping in the deringing filter. It corresponds to the variable named cdef\_damping\_minus\_3 from the specification.

**y\_strengths[]**

Specifies the primary and secondary filter strengths for the y-channel. This corresponds to the cdef\_y\_pri\_strength, and cdef\_y\_sec\_strength syntax elements from the specification.

**y\_strengths::primary**

Specifies the primary filter strength for the y-channel.

**y\_strengths::secondary**

Specifies the secondary filter strength for the y-channel.

**uv\_strengths**

Specifies the primary and secondary filter strengths for the uv-channels. This corresponds to the cdef\_uv\_pri\_strength, and cdef\_uv\_sec\_strength syntax elements from the specification.

**uv\_strengths::primary**

Specifies the primary filter strength for the uv-channels.

**uv\_strengths::secondary**

Specifies the secondary filter strength for the uv-channels.

**bits**

Specifies the number of bits used to signal the CDEF filter setting. It corresponds to the syntax element named cdef\_bits from the specification.

### 4.2.8 Quantization Parameters

**base\_qindex**

Indicates the base frame qindex. It corresponds to the syntax element named base\_q\_idx from the specification.

**y\_dc\_delta\_q, u\_dc\_delta\_q, v\_dc\_delta\_q**

Specify the correspond Y, U and V DC quantizer relative to base\_qindex. These correspond to the variables: DeltaQYDc, DeltaQUDc, and DeltaQVDc from the specification.

**u\_ac\_delta\_q, v\_ac\_delta\_q**

Specify the correspond U and V AC quantizer relative to base\_qindex. These correspond to the variables: DeltaQUAc and DeltaQVAc from the specification.

**delta\_q\_res**

Specifies the left shift which should be applied to decoded quantizer index delta values. It corresponds to the syntax element of the same name from the specification.

**delta\_q\_present**

Specifies weather the quantizer index delta values are presnt. It corresponds to the syntax element of the same name from the specification.

**qm\_y, qm\_u, qm\_v**

Specify the level in the quantizer matrix that should be used for luma plane, U-plane, and V-plane decoding, respectively. When unspecified (using\_qmatrix=0), these values will be 0xFF (which is an invalid quantizer matrix level). These correspond to the syntax elements of the same name from the specification.

### 4.2.9 Segmentation Parameters

**enabled**

When equal to 1, indicates that this frame makes use of the segmentation tool. It corresponds to the syntax element named segmentation\_enabled from the specification.

**update\_map**

When equal to 1, indicates that the segmentation map is updated during the decoding of this frame. It corresponds to the syntax element named segmentation\_update\_map from the specification.

**update\_data**

When equal to 1, indicates that new parameters are supplied for each segment. It corresponds to the syntax element named segmentation\_update\_data from the specification.

**temporal\_update**

When equal to 1, indicates that the updates to the segmentation map are coded relative to the existing segmentation map. It corresponds to the syntax element named segmentation\_temporal\_update from the specification.

**Reserved**

Reserved bit fields to complete the packed structure. Shall be set to 0. The accelerator shall ignore the values in the reserved bit fields.

**ControlFlags**

Provides an alternative way to access the bit fields related to segmentation flags.

**feature\_mask**

Specifies which segment data features are updated in this frame. It corresponds to the syntax element feature\_enabled from the specification.

**feature\_data**

Specifies segmentation feature values. It corresponds to the syntax element feature\_value from the specification.

### 4.2.10 Film Grain Parameters

**apply\_grain**

When equal to 1, indicates that film grain should be added to this frame. It corresponds to the syntax element of the same name from the specification.

**scaling\_shift\_minus8**

Specifies the shift -8 applied to values of the chroma component. It corresponds to the syntax element named grain\_scaling\_minus8 from the specification.

**chroma\_scaling\_from\_luma**

When equal to 1, specifies that the chroma scaling is inferred from the luma scaling. It corresponds to the syntax element of the same name from the specification.

**ar\_coeff\_lag**

Specifies the number of auto-regressive coefficients for luma and chroma. It corresponds to the syntax element of the same name from the specification.

**ar\_coeff\_shift\_minus6**

Specifies the range of auto-regressive coefficients. It corresponds to the syntax element of the same name from the specification.

**grain\_scale\_shift**

Specifies how much the Gaussian random numbers should be scaled down during the grain synthesis process. It corresponds to the syntax element of the same name from the specification.

**overlap\_flag**

When equal to 1, indicates that the overlap between film grain blocks shall be applied. It corresponds to the syntax element of the same name from the specification.

**Reserved**

Reserved bit fields to complete the packed structure. Shall be set to 0. The accelerator shall ignore the values in the reserved bit fields.

**ControlFlags**

Provides an alternative way to access the bit fields related to film grain flags.

**grain\_seed**

Specifies the starting value for the pseudo-random number generator used for film grain synthesis. This corresponds to the syntax element of the same name from the specification.

**scaling\_points\_y, scaling\_points\_cb, scaling\_points\_cr**

Represents the x,y coordinates for the piecewise linear scaling function for each plane: y, cb and cr. This is a 2D array, at each piecewise segment the first value specified is the x (luma value) coordinate and the second value is the scaling value (y, output). These correspond to the point\_y\_value, point\_y\_scaling, point\_cb\_values, point\_cb\_scaling, point\_cr\_value and point\_cr\_scaling syntax elements from the specification.

**num\_y\_points, num\_cb\_points, num\_cr\_points**

Specifies the number of valid piecewise segments specified in scaling\_points\_y, scaling\_points\_cb, and scaling\_points\_cr. These correspond to the syntax elements of the same name from the specification.

**ar\_coeffs\_y, ar\_coeffs\_cb, ar\_coeffs\_cr**

Specifies the auto-regressive coefficients for the Y, U and V planes. These correspond to the ar\_coeffs\_y\_plus\_128, ar\_coeffs\_cb\_plus\_128 and ar\_coeffs\_cr\_plus\_128 syntax elements from the specification.

**cb\_mult, cr\_mult**

Specifies the multiplier for the Cb and Cr components used in derivation of the input index to the component scaling function. These correspond to the syntax elements of the same name from the specification.

**cb\_luma\_mult, cr\_luma\_mult**

Specifies the multiplier used for the luma component into the Cb and Cr component scaling functions. These correspond to the syntax elements of the same name from the specification.

**cb\_offset, cr\_offset**

Specifies the offset used in derivation of the input index to the Cb and Cr component scaling functions. These correspond to the syntax elements of the same name from the specification.

### 4.2.11 Miscellaneous Parameters

**interp\_filter**

Corresponds to the same syntax element of the same name in the specification and affects the decoding process of motion compensation interpolation accordingly. The table below shows the possible values of interp\_filter.

|  |  |
| --- | --- |
| Value | Filter type |
| 0 | normal 8-tap |
| 1 | smooth 8-tap |
| 2 | sharp 8-tap |
| 3 | bilinear |
| 4 | all filters |

**Reserved32Bits**

Field reserved for future use.

**StatusReportFeedbackNumber**

Arbitrary number set by the host decoder to use as a tag in the status report feedback data. The value should not be equal to 0, and should be different in each call to **Execute**. For more information, see section 6 (Status Report Data Structure).

***Header Inclusion Requirements***

**Header:** Include dxva.h.

# 5. Tile Control Data Structure

The tile control structure is always provided with a bitstream. The total quantity of data in the bitstream buffer (and the amount of data reported by the host software decoder) shall be an integer multiple of 128 bytes. This structure shall always contain information about all tiles necessary to decode the frame, and nothing else (for example, only tiles in the current OperatingPointIDC will be included). The bitstream may have more data than as specified by this structure, and may have data between tiles. The accelerator should ignore this extra data (the decoder may opt to send the entire contents of the bitstream including OBU headers, but must specify offsets to tile data).

## 5.1 Syntax

The **DXVA\_Tile\_AV1** data structure, is sent by the host software decoder to the accelerator to convey tile data locations. This structure is sent as part of the SLICE\_CONTROL argument type (D3D12\_VIDEO\_DECODE\_ARGUMENT\_TYPE\_SLICE\_CONTROL, D3D11\_VIDEO\_DECODER\_BUFFER\_SLICE\_CONTROL, or DXVA2\_SliceControlBufferType). The buffer associated with this argument must have at least one **DXVA\_Tile\_AV1** structure but may also have an array of multiple **DXVA\_Tile\_AV1** structures to specify more than one tile. The number of tiles can be inferred by dividing the size of the buffer by the size of the **DXVA\_Tile\_AV1** structure.

The form of this data structure is shown below:

typedef struct \_DXVA\_Tile\_AV1 {

UINT DataOffset;

UINT DataSize;

USHORT row;

USHORT column;

USHORT Reserved16Bits;

UCHAR anchor\_frame;

UCHAR Reserved8Bits;

} DXVA\_Tile\_AV1, \*LPDXVA\_Tile\_AV1;

## 5.2 Semantics

**DataOffset**

If this member locates the compressed bitstream data for the current tile. Specifically this is the byte offset from the start of the bitstream buffer of either **tile\_list\_obu**, **codec\_tile\_data** or **tile\_group\_obu**, **init\_symbol( tileSize )** from the specification.

**DataSize**

Number of bytes in the bitstream data buffer that are associated with this tile or list of tiles starting at the byte given by **DataOffset**.

**anchor\_frame**

Specifies an associated **anchor\_frame** for tile list OBU. This is to be translated to the the source frame buffer/surface index for the reference tile picture, from the anchor\_frame\_idx value in the specification. If this tile is not part of a Tile List OBU then this frame will have the standard invalid frame index of 255.

**row**

Specifies the row that this tile belongs to. When part of a Tile List OBU this specifies the **anchor\_tile\_row** from the specification.

**column**

Specifies the column that this tile belongs to. When part of a Tile List OBU this specifies the **anchor\_tile\_column** from the specification.

# 6. Status Report Data Structure

The **DXVA\_Status\_AV1** data structure is sent by the accelerator to the host software decoder to convey decoding status information. This structure is used when bDXVA\_Func is 7.

The status reporting command does not use a compressed buffer. Instead, the host software decoder provides a buffer as private output data. For more information, see section 1.7 (Status Reporting) of this specification.

The status information command should be asynchronous to the decoding process. The host software decoder should not wait to receive status information on a process before it proceeds to initiate another process. After the host software decoder has received a status report for a particular operation, the accelerator shall discard that information and not report it again. (That is, the results of each particular operation shall not be reported to the host software decoder more than once.) Accelerators shall be capable of providing status information for every buffer for every operation performed.

Accelerators are required to store at least a minimum of 512 **DXVA\_Status\_AV1** structures internally, pending status requests from the host software decoder. An accelerator may (and should) exceed this storage capacity. If the accelerator discards reporting information, it should discard the oldest data first. The accelerator should provide status reports in approximately reverse temporal order of when the operations were completed. That is, status reports for the most recently completed operations should appear earlier in the list of status report data structures.

**Note –** As previously stated, the term *should* describes guidelines that are encouraged but are not mandatory requirements.

## 6.1 Syntax

The **DXVA\_Status\_AV1** data structure is sent by the accelerator to the host software decoder to convey decoding status information. The data structure and associated semantics are essentially the same as for the previous DXVA\_Status\_H264, DXVA\_Status\_HEVC, and DXVA\_Status\_VPx data structures. It has been given a new name so that the data structures used for AV1 will have names that are associated with the new design. For convenience, the form of this data structure is shown below:

typedef struct \_DXVA\_Status\_AV1 {

UINT StatusReportFeedbackNumber;

DXVA\_PicEntry\_AV1 CurrPic;

UCHAR bBufType;

UCHAR bStatus;

UCHAR bReserved8Bits;

USHORT wNumMbsAffected;

} DXVA\_Status\_AV1, \*LPDXVA\_Status\_AV1;

## 6.2 Semantics

**StatusReportFeedbackNumber**

Contains the value of **StatusReportFeedbackNumber** set by the host software decoder in the picture parameters data structure for the associated operation.

**CurrPic**

Specifies the uncompressed destination surface that was affected by the operation.

**bBufType**

Indicates the type of compressed buffer associated with this status report. If **bStatus** is 0, the value of **bBufType** may be 0xFF. This value indicates that the status report applies to all of the compressed buffers conveyed in the associated **Execute** call. Otherwise, if **bBufType** is not 0xFF, it must contain one of the following values, defined in dxva.h:

|  |  |
| --- | --- |
| **Value** | **Description** |
| DXVA\_PICTURE\_DECODE\_BUFFER (1) | Picture decoding parameter buffer. |
| DXVA\_SLICE\_CONTROL\_BUFFER (6) | Slice control buffer. |
| DXVA\_BITSTREAM\_DATA\_BUFFER (7) | Bitstream data buffer. |

**bStatus**

Indicates the status of the operation as shown in the table below.

|  |  |
| --- | --- |
| **Value** | **Description** |
| 0 | The operation succeeded. |
| 1 | Minor problem in the data format. The host decoder should continue processing. |
| 2 | Significant problem in the data format. The host decoder may continue executing or skip the display of the output picture. |
| 3 | Severe problem in the data format. The host decoder should restart the entire decoding process, starting at a sequence or random-access entry point. |
| 4 | Other severe problem. The host decoder should restart the entire decoding process, starting at a sequence or random-access entry point. |

If the value is 3 or 4, the host software decoder should halt the decoding process unless it can take corrective action.

**bReserved8Bits**

This structure member has no meaning, and the value shall be 0. The accelerator shall ignore its value.

**wNumMbsAffected**

If **bStatus** is not 0, this member contains the accelerator's estimate of the number of super-blocks in the decoded frame that were adversely affected by the reported problem. If the accelerator does not provide an estimate, the value is 0xFFFF.

If **bStatus** is 0, the accelerator may set wNumMbsAffected to the number of super-blocks that were successfully decoded by the operation. If the accelerator does not provide an estimate, it shall set the value either to 0 or to 0xFFFF.

***Header Inclusion Requirements***

**Header:** Include dxva.h.

# 7. Restricted-Mode Profiles

The following restricted-mode profiles for DXVA operations for AV1 video decoding are defined. The GUIDs that identify these profiles will be defined in the header file dxva.h. Additional restricted-mode profiles for DXVA operations may be defined in the future.

## 7.1 DXVA\_ModeAV1\_VLD\_Profile0 Profile

This profile supports the features necessary for a decoder that conforms to AV1 profile 0. In this profile, the accelerator performs bitstream parsing, inverse quantization scaling, inverse transform processing, motion compensation, and deblocking with the support of 4:2:0 chroma subsampling and monochrome.

All data buffers shall contain only data that is consistent with the constraints specified for AV1 profile 0.

The associated GUID definition for the corresponding entry in the dxva.h header file is as follows:

DEFINE\_GUID(DXVA\_ModeAV1\_VLD\_Profile0, 0xb8be4ccb, 0xcf53, 0x46ba, 0x8d, 0x59, 0xd6, 0xb8, 0xa6, 0xda, 0x5d, 0x2a);

This profile should support the following output texture formats:

* DXGI\_FORMAT\_NV12 for 8-bit YUV 4:2:0
* DXGI\_FORMAT\_P010 for 10-bit YUV 4:2:0

This profile may additionally support the following output texture formats:

* DXGI\_FORMAT\_R8\_UNORM for 8-bit monochrome
* DXGI\_FORMAT\_R16\_UNORM for 10-bit monochrome

## 7.2 DXVA\_ModeAV1\_VLD\_Profile1 Profile

This profile supports the features necessary for a decoder that conforms to AV1 profile 1. In this profile, the accelerator performs bitstream parsing, inverse quantization scaling, inverse transform processing, motion compensation, and deblocking with the support of 4:4:4 chroma subsampling.

All data buffers shall contain only data that is consistent with the constraints specified for AV1 profile 1.

The associated GUID definition for the corresponding entry in the dxva.h header file is as follows:

DEFINE\_GUID(DXVA\_ModeAV1\_VLD\_Profile1, 0x6936ff0f, 0x45b1, 0x4163, 0x9c, 0xc1, 0x64, 0x6e, 0xf6, 0x94, 0x61, 0x08);

This profile should support the following output texture formats:

* DXGI\_FORMAT\_AYUV for 8-bit YUV 4:4:4
* DXGI\_FORMAT\_Y410 for 10-bit YUV 4:4:4

## 7.3 DXVA\_ModeAV1\_VLD\_Profile2 Profile

This profile supports the features necessary for a decoder that conforms to AV1 profile 2, at only 8 and 10 bits, excluding 12 bits. In this profile, the accelerator performs bitstream parsing, inverse quantization scaling, inverse transform processing, motion compensation, and deblocking with the support of 4:2:0, 4:2:2 and 4:4:4 chroma subsampling and monochrome.

All data buffers shall contain only data that is consistent with the constraints specified for AV1 profile 2, except 12-bit bitstreams.

The associated GUID definition for the corresponding entry in the dxva.h header file is as follows:

DEFINE\_GUID(DXVA\_ModeAV1\_VLD\_Profile2, 0x0c5f2aa1, 0xe541, 0x4089, 0xbb, 0x7b, 0x98, 0x11, 0x0a, 0x19, 0xd7, 0xc8);

This profile should support the following output texture formats:

* DXGI\_FORMAT\_YUY2 for 8-bit YUV 4:2:2
* DXGI\_FORMAT\_Y210 for 10-bit YUV 4:2:2
* DXGI\_FORMAT\_R8\_UNORM for 8-bit monochrome
* DXGI\_FORMAT\_R16\_UNORM for 10-bit monochrome

## 7.4 DXVA\_ModeAV1\_VLD\_12bit\_Profile2 Profile

This profile supports the features necessary for a decoder that conforms to AV1 profile 2, at 8, 10 and 12 bits. In this profile, the accelerator performs bitstream parsing, inverse quantization scaling, inverse transform processing, motion compensation, and deblocking with the support of 4:2:0, 4:2:2 and 4:4:4 chroma subsampling and monochrome.

All data buffers shall contain only data that is consistent with the constraints specified for AV1 profile 2, including support for 12-bit bitstreams.

The associated GUID definition for the corresponding entry in the dxva.h header file is as follows:

DEFINE\_GUID(DXVA\_ModeAV1\_VLD\_12bit\_Profile2, 0x17127009, 0xa00f, 0x4ce1, 0x99, 0x4e, 0xbf, 0x40, 0x81, 0xf6, 0xf3, 0xf0);

This profile should support the following output texture formats:

* DXGI\_FORMAT\_YUY2 for 8-bit YUV 4:2:2
* DXGI\_FORMAT\_Y210 for 10-bit YUV 4:2:2
* DXGI\_FORMAT\_P016 for 12-bit YUV 4:2:0
* DXGI\_FORMAT\_Y216 for 12-bit YUV 4:2:2
* DXGI\_FORMAT\_Y416 for 12-bit YUV 4:4:4
* DXGI\_FORMAT\_R8\_UNORM for 8-bit monochrome
* DXGI\_FORMAT\_R16\_UNORM for 10-bit monochrome
* DXGI\_FORMAT\_R16\_UNORM for 12-bit monochrome

## 7.5 DXVA\_ModeAV1\_VLD\_12bit\_Profile2\_420 Profile

This profile supports the features necessary for a decoder that conforms to AV1 profile 2, at 8, 10 and 12 bits. In this profile, the accelerator performs bitstream parsing, inverse quantization scaling, inverse transform processing, motion compensation, and deblocking with the support of 4:2:0 chroma subsampling and monochrome.

All data buffers shall contain only data that is consistent with the constraints specified for AV1 profile 2, including support for 12-bit bitstreams.

The associated GUID definition for the corresponding entry in the dxva.h header file is as follows:

DEFINE\_GUID(DXVA\_ModeAV1\_VLD\_12bit\_Profile2\_420, 0x2d80bed6, 0x9cac, 0x4835, 0x9e, 0x91, 0x32, 0x7b, 0xbc, 0x4f, 0x9e, 0xe8);

This profile should support the following output texture formats:

* DXGI\_FORMAT\_P016 for 12-bit YUV 4:2:0
* DXGI\_FORMAT\_R8\_UNORM for 8-bit monochrome
* DXGI\_FORMAT\_R16\_UNORM for 10-bit monochrome
* DXGI\_FORMAT\_R16\_UNORM for 12-bit monochrome

# 8. For More Information

* DirectX Video Acceleration 2.0 documentation: <http://go.microsoft.com/fwlink/?LinkId=94771>

Web addresses can change, so you might be unable to connect to the Web site or sites mentioned here.