DirectX Video Acceleration Specification for VP8 and VP9 Video Coding

August 2016

Srinath Reddy, Yongjun Wu and Gary J. Sullivan

Abstract – This document contains a specification for support of VP8 and VP9 video decoding (according to the available VP8 and VP9 source code and public documents) within the Microsoft Windows DirectX Video Acceleration (DXVA) API/DDI context. This specification includes support of the VP8 coding format,VP9 Profile 0 (for 8 bit 4:2:0 video) and VP9 Profile 2 in 10 bit mode. The document describes high-level design concepts and specific VP8 and VP9 extensions to DXVA interfaces and data structures of VP8 and VP9 video decoding. This document specifies only off-host Variable Length Decoding (VLD) profiles for VP8 and VP9 video decoding.

Contents

1. Introduction

This specification defines extensions to DirectX® Video Acceleration (DXVA) to support decoding of VP8 and VP9 video. See, e.g., https://en.wikipedia.org/wiki/VP8 and http://en.wikipedia.org/wiki/VP9#cite_note-VP9FinalizationMay2013GoogleGroup-11.

This specification assumes the reader is familiar with the VP8 and VP9 source code and the associated publicly available documents, and with the basic design of DXVA. This specification of DXVA usage for VP8 and VP9 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 VP8 and VP9 video decoding. We do not plan to specify VP8 and VP9 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 functionality 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 (although trick play functionality in this case is somewhat limited by the design of the entropy decoding process, which has a stored internal state).

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.

1.1 *Referenced Documents and Software*

There is a published IETF RFC 6386 document that describes the VP8 video format and decoding process in some detail (although it is not an entirely complete specification):

https://tools.ietf.org/html/rfc6386

However, currently there are only high-level overviews about VP9 video coding that are publicly available, such as the following:

http://forum.doom9.org/showthread.php?t=168947 *http://tools.ietf.org/html/draft-grange-vp9-bitstream-00*

http://files.meetup.com/9842252/Overview-VP9.pdf

http://community.roxen.com/developers/idocs/drafts/draft-grange-vp9-bitstream-00.html

Associated open-source reference software for VP8 and VP9 is available within the libvpx library at the following links:

https://code.google.com/p/webm/downloads/list

https://code.google.com/p/webm/source/browse/vp9/?repo=libvpx&r=7b8dfcb5a2cfb01ee7a6009d945d06559b564d06

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 VP8 and VP9 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 VP9 bitstreams of Profile 0 and Profile 2 in 10-bit mode, and VP8 bitstreams. Neither VP8 or VP9 include support for interlaced-scan field-based coding and field-based display – they are 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 VP8 and VP9 video decoding; only off-host VLD mode of DXVA operation is specified for VP8 and VP9 video decoding.

1.4 Picture Data

The following data must be conveyed for each VP8 or VP9 frame in order to decode each frame independently without serial dependencies or with minimized serial dependencies. For simplicity, the same or similar flag names from VP8 or VP9 reference software are used. For further details, see section 3 (VP9 Picture Parameters Data Structure) and section 4 (VP8 Picture Parameters Data Structure) of this specification.

1.4.1 VP9 Picture Data

VP9 picture-level data includes the following:

- Basic coding parameters and dimensions, including

    - profile
    - width and height
- Frame buffering state and reference list related information, including:
    - CurrPic (indicating the current destination surface)
    - frame_type
    - ref_frame_map[]
    - ref_frame_coded_width[]
    - ref_frame_coded_height[]
    - frame_refs[]
    - ref_frame_sign_bias[]
- Flags and associated data controlling particular coding features that are the same for the whole frame, including
    - error_resilient_mode
    - intra_only
    - reset_frame_context
    - allow_high_precision_mv
    - interp_filter
    - refresh_frame_context
    - frame_parallel_decoding_mode
    - frame_context_idx
- Syntax element values for deblocking, quantization and tile partition, including
    - filter_level
    - sharpness_level
    - mode_ref_delta_enabled
    - ref_deltas[]
    - mode_deltas[]
    - base_qindex
    - y_dc_delta_q
    - uv_dc_delta_q
    - uv_ac_delta_q
    - log2_tile_cols
    - log2_tile_rows
- Syntax element values for the segmentation map, including:
    - enabled
    - tree_probs[]
    - pred_probs[]
    - abs_delta
    - feature_mask[]
    - feature_data[][]

1.4.2 VP8 Picture Data

VP8 picture-level data includes the following:

- Basic coding parameters and dimensions, including
  - version
  - width and height
- Frame buffering state and reference list related information, including:
  - CurrPic (indicating the current destination surface)
  - frame_type
  - alt_fb_idx
  - gld_fb_idx
  - lst_fb_idx
  - ref_frame_sign_bias_golden
  - ref_frame_sign_bias_altref
- Flags and associated data controlling particular coding process that are the same for the whole frame, including
  - clamp_type
  - refresh_entropy_probs
  - vp8_coef_update_probs[][][][]
  - mb_no_coeff_skip
  - prob_skip_false
  - prob_intra
  - prob_last
  - prob_golden
  - intra_16x16_prob[]
  - intra_chroma_prob[]
  - vp8_mv_update_probs[][]
- Syntax element values for deblocking, quantization and bitstream partition, including
  - filter_type
  - filter_level
  - sharpness_level
  - mode_ref_delta_enabled
  - mode_ref_lf_delta_update
  - ref_lf_deltas []
  - mode_lf_deltas[]
  - base_qindex
  - y1dc_delta_q
  - y2dc_delta_q
  - y2ac_delta_q
  - uvdc_delta_q
  - uvac_delta_q
  - log2_nbr_of_dct_partitions

- Syntax element values for the segmentation map, including:
  - update_mb_segmentation_map
  - update_mb_segmentation_data
  - mb_segement_abs_delta
  - segment_feature_data [][]
  - mb_segment_tree_probs []

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.
- One slice control buffer.
- One or more bitstream data buffers.

VP8 and VP9 do not have multiple-slice support. Hence there is always one and only one slice control buffer per compressed frame.

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 VP8 and VP9 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 or more times, sending one or more compressed data buffers to the accelerator and specifying the operations to perform on the buffers. 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 VP8 and VP9 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 VP8 and VP9 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 slice control data buffers.
- One or more bitstream data buffers.

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_VPx**).

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 VP9 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.

Specifically, the accelerator shall store the set of information necessary for use in VP9 inter-picture prediction along with each decoded reference picture, such as the co-located motion vectors from a decoded reference frame used in the motion vector candidate list. It also needs to accumulate counts for various symbols actually decoded over a frame used for backward context updates on the completion of current frame decoding.

VP8 video decoding only uses one reference frame, either the previous frame (last frame), the golden frame or the altref frame. Co-located motion vectors from a decoded reference frame are not employed during decoding. Instead, only spatial neighbor motion vectors are used. VP8 video coding uses forward probability updates without backward context updates. The forward probability updates are persistent. That is, a probability updated on one frame is in effect for all subsequent frames until the next key frame (an I frame that resets inter-picture prediction processing), or until the probability is explicitly updated by another frame.

In VP9, each inter frame might be coded at a different resolution than the previous frame(s). When creating inter predictions, the reference frame needs to be scaled up or down accordingly. The scaling filters are 16th-pel accurate and use 8-tap filters. The scaling on reference frames shall be done by the accelerator as an internal operation when necessary. If the maximum coded resolution is known for VP9 video decoding, host decoder may allocate the surfaces at the maximum coded resolution, and accelerator shall be able to decode pictures at lower resolutions into the surfaces at the maximum coded resolution, starting from top-left corner and taking care of surface stride at the maximum coded resolution and decoding height and width at lower resolutions with the chroma offset surface size based instead of coded picture size based. Host decoder shall output the pictures at lower resolutions decoded in the surfaces at the maximum coded resolution with proper cropping window.

In VP8, coded resolution changes are only allowed to happen at key frames. Inter-prediction from a reference frame at a different coded resolution from the resolution of the current coded frame is not allowed. Hence, in VP8, there is no need to perform scaling on reference frames for motion vector prediction and motion compensation.

1.9 Configuration Parameters

This section describes the configuration parameters for VP8 and VP9 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 VP8 and VP9 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_VPx_Short** structure for VP8 and VP9 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

Shall be 0 except for any bits that may be set by the Accelerator to signal Decoder Specific Support. These are described in Section 1.9.3.

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 decoding resolution (picture width or height) has changed or that the accelerator capability requirements have changed, such as enabling or disabling downsampling of the output.

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.

Bit 13 is reserved for future use and shall be set to 0

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

2. DXVA_PicEntry_VPx Data Structure

The **DXVA_PicEntry_VPx** structure specifies a reference to an uncompressed surface. It is used in other data structures described in this document. The data structure itself is the same as the previous DXVA_PicEntry_H264 and DXVA_PicEntry_HEVC data structures. It has been given a new name so that the data structures used for VP8 and VP9 will have names that are associated with the new design. For convenience, the form of this data structure is shown below.

2.1 Syntax

```
typedef struct _DXVA_PicEntry_VPx {
  union {
      struct {
           UCHAR Index7Bits     : 7;
           UCHAR AssociatedFlag : 1;
      };
      UCHAR bPicEntry;
  };
} DXVA_PicEntry_VPx, *LPDXVA_PicEntry_VPx;
```

2.2 Semantics

Index7Bits

An index that identifies an uncompressed surface for the **CurrPic** or **ref_frame_map[]** and **frame_refs[]** members of the picture parameters structure in VP9, and for the **CurrPic** or **alt_fb_idx, gld_fb_idx,** and **lst_fb_idx** members of the picture parameters structure in VP8.

When Index7Bits is used in those members of the picture parameters structure, the value directly specifies the DXVA index of an uncompressed surface.

When **Index7Bits** does not contain an index to a valid uncompressed surface, the value shall be set to 127, to indicate that the index is invalid.

AssociatedFlag

Shall be 0 when **Index7Bits** is valid. When **Index7Bits** is equal to 127 (indicating that it does not contain a valid index), it shall be 1.

bPicEntry

Accesses the entire 8 bits of the union. Equal to 0xFF when it does not contain a valid index.

3. VP9 Picture Parameters Data Structure

The **DXVA_PicParams_VP9** structure provides the picture-level parameters of a compressed picture for VP9 video decoding. This structure is used for VP9 when bDXVA_Func is 1 and the buffer type is DXVA2_PictureParametersBufferType (in DXVA 2.0).

3.1 Syntax

```
typedef struct _segmentation_VP9 {
     union {
       struct {
           UCHAR      enabled                         : 1;
           UCHAR      update_map                      : 1;
           UCHAR      temporal_update                 : 1;
           UCHAR      abs_delta                       : 1;
           UCHAR      ReservedSegmentFlags4Bits       : 4;
       };
       UCHAR    wSegmentInfoFlags;
     };
  UCHAR tree_probs[7];
  UCHAR pred_probs[3];
  SHORT feature_data[8][4];
  UCHAR feature_mask[8];
} DXVA_segmentation_VP9;


typedef struct _DXVA_PicParams_VP9 {
     DXVA_PicEntry_VPx     CurrPic;
     UCHAR                 profile;
     union {
       struct {
           USHORT     frame_type                        : 1;
           USHORT     show_frame                        : 1;
           USHORT     error_resilient_mode              : 1;
           USHORT     subsampling_x                     : 1;
           USHORT     subsampling_y                     : 1;
           USHORT     extra_plane                       : 1;
           USHORT     refresh_frame_context             : 1;
           USHORT     frame_parallel_decoding_mode      : 1;
           USHORT     intra_only                        : 1;
           USHORT     frame_context_idx                 : 2;
           USHORT     reset_frame_context               : 2;
           USHORT     allow_high_precision_mv           : 1;
           USHORT     ReservedFormatInfo2Bits           : 2;
       };
       USHORT   wFormatAndPictureInfoFlags;
     };
     UINT       width;
     UINT       height;
     UCHAR      BitDepthMinus8Luma;
     UCHAR      BitDepthMinus8Chroma;
     UCHAR      interp_filter;
     UCHAR      Reserved8Bits;
     DXVA_PicEntry_VPx     ref_frame_map[8];
     UINT       ref_frame_coded_width[8];
     UINT       ref_frame_coded_height[8];
     DXVA_PicEntry_VPx     frame_refs[3];
     CHAR       ref_frame_sign_bias[4];
     CHAR       filter_level;
     CHAR       sharpness_level;
     union {
       struct {
           UCHAR      mode_ref_delta_enabled          : 1;
           UCHAR      mode_ref_delta_update           : 1;
           UCHAR      use_prev_in_find_mv_refs        : 1;
           UCHAR      ReservedControlInfo5Bits        : 5;
       };
       UCHAR    wControlInfoFlags;
     };
     CHAR       ref_deltas[4];
     CHAR       mode_deltas[2];
     SHORT      base_qindex;
     CHAR       y_dc_delta_q;
     CHAR       uv_dc_delta_q;
     CHAR       uv_ac_delta_q;
     DXVA_segmentation_VP9  stVP9Segments;
     UCHAR      log2_tile_cols;
     UCHAR      log2_tile_rows;
     USHORT     uncompressed_header_size_byte_aligned;
     USHORT     first_partition_size;
     USHORT     Reserved16Bits;
     UINT       Reserved32Bits;
     UINT       StatusReportFeedbackNumber;
   } DXVA_PicParams_VP9, *LPDXVA_PicParams_VP9;
```

3.2 Semantics

CurrPic

Specifies the destination frame buffer/surface index for the decoded picture. In this context, the **AssociatedFlag** has no meaning and shall be 0, and the accelerator shall ignore its value.

profile

Indicates the profile of the VP9 bitstream. The VP9 video coding format defines four profiles: profile 0, profile 1, profile 2, and profile 3. Profile 0 supports 4:2:0 chroma sampling with 8 bits per sample. Profile 1 adds support for 4:2:2 and 4:4:4 chroma sampling, alpha channels, and depth channels. VP9 was also later extended by adding two higher bit-depth profiles: profile 2 and profile 3. Profile 2 supports bit depths of 10 to 12 bits per sample with 4:2:0 chroma sampling. Profile 3 supports 4:2:2 and 4:4:4 chroma sampling and alpha channels.

For current purposes, only Profile 0 (for 8 bit video) and Profile 2 restricted to 10 bit only are supported for VP9 by DXVA decoding profiles specification (see clause 7, Restricted-Mode Profiles).

frame_type

Specifies the frame type of the current frame. It corresponds to the syntax element of the same name in the reference software and affects the decoding process accordingly. The allowed values are 0 and 1, for two types of VP9 frames, KEY_FRAME and INTER_FRAME.

show_frame

Indicates whether the current frame is intended to be output and displayed after its decoding is completed. In DXVA, this has no direct effect, as the host controls the display of decoded frames separately by other function calls.

error_resilient_mode

VP9 has a frame level error_resilient_mode flag. When the flag is turned on, it supports a coding mode that allows decoding to continue (although with error propagation) even when some frames are lost. In particular, the following modifications of the decoding process apply in error resilient mode:

1. The entropy coding context probabilities are reset to defaults at the beginning of each frame. (This prevents propagation of forward updates as well as backward updates),
2. For MV reference selection, the co-located MV from a previously encoded reference frame can no longer be included in the reference candidate list,
3. For MV reference selection, the sorting of the initial list of motion vector reference candidates based on a search in the reference frame buffer is disabled.

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

extra_plane

Indicates whether an alpha channel or depth channel is present. The allowed values are restricted by the **profile** value.

refresh_frame_context, frame_parallel_decoding_mode

Indicate whether the frame context state for the entropy decoding process is refreshed or not, when error_resilient_mode flag is equal to 0, as shown in the table below.

refresh_frame_context	**frame_parallel_decoding_mode**	action done
1	0	refresh the context by backward update
0	0	do not refresh
0	1	do not refresh
1	1	refresh the context by forward update

When error_resilient_mode flag is equal to 1, refresh_frame_context shall be equal to 0 and frame_parallel_decoding_mode shall be equal to 1.

intra_only, frame_context_idx, reset_frame_context

If frame_type is equal to 0 (KEY_FRAME), or error_resilient_mode is equal to 1, all frame contexts are reset. Only when intra_only is equal to 1, frame contexts are set as specified by reset_frame_context as follows.

If reset_frame_context is equal to 0 or 1, do not reset any contexts.

If reset_frame_context is equal to 2, reset the context specified in the frame header by frame_context_idx.

If reset_frame_context is equal to 3, reset all frame contexts.

allow_high_precision_mv

Corresponds to the same syntax element of the same name in the reference software and affects the decoding process of motion vector and motion compensation accordingly. When frame_type is equal to 0 (KEY_FRAME), allow_high_precision_mv shall be equal to 0.

ReservedFormatInfo2Bits

Reserved bit fields which shall be set to 0. The accelerator shall ignore the values in the reserved bit fields.

wFormatAndPictureInfoFlags

Provides an alternative way to access the bit fields.

width, height

Specify the coded width and height of the current frame. These correspond to the syntax elements of the same names in the reference software and affect the decoding process accordingly. Each VP9 frame may be coded at a different resolution than the previous frame(s).

BitDepthMinus8Luma, BitDepthMinus8Chroma

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

interp_filter

Corresponds to the same syntax element of the same name in the reference software and affects the decoding process of motion compensation interpolation accordingly. The motion compensation filter in VP9 has 1/8th sample position precision. 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	switchable

Reserved8Bits

Reserved bit fields for 32-bit alignment. Shall be set to 0. The accelerator shall ignore the values in the reserved bit fields.

ref_frame_map[]

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 **bPicEntry** to 0xFF. If **bPicEntry** is 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 **ref_frame_map[]** 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 **ref_frame_map[]** array.

The **AssociatedFlag** has no meaning and shall be 0, and the accelerator shall ignore its value.

Note – The accelerator must use the content of the **ref_frame_map[]** 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).

ref_frame_coded_width[] and **ref_frame_coded_height[]**

Indicate the coded width and height of the corresponding reference frames with the same indices in **ref_frame_map[]**. If the corresponding entry in **ref_frame_map[]** is not used for decoding the current picture, or any subsequent pictures, the entry in **ref_frame_coded_width[]** and **ref_frame_coded_height[]** should be set to 0.

frame_refs[]

Indicates the reference surfaces to be used for inter prediction (known as Last, Golden and AltRef) during current frame decoding. The reference surface indices in **frame_refs[]** shall exist in **ref_frame_map[]** array.

Note – The VP9 decoder maintains a pool (**ref_frame_map[]**) of 8 reference pictures at all times. Each frame picks 3 reference frames (**frame_refs[]**) from the pool to use for inter prediction (known as Last, Golden, and AltRef) 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 (**ref_frame_map[]**), 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.

Each new inter frame can be coded using a different resolution from that of the previous frame. When creating inter predictions, the reference frame is scaled up or down accordingly when necessary. The scaling filters are 16th-pel accurate and 8-tap. The scaling on reference frames shall be done by accelerator when necessary.

ref_frame_sign_bias[]

Corresponds to the same syntax element of the same name in the reference software and affects the decoding process about reference mode and reference selection accordingly. **ref_frame_sign_bias[]** affects the motion vector candidate list, the setup of compound reference mode and compound motion compensation. The enums and indices for **ref_frame_sign_bias[]** are defined in the same way as the enum MV_REFERENCE_FRAME structure in reference software mentioned previously, where INTRA_FRAME = 0, LAST_FRAME = 1, GOLDEN_FRAME = 2, and ALTREF_FRAME = 3.

filter_level, sharpness_level

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

mode_ref_delta_enabled, mode_ref_delta_update, ref_deltas[], mode_deltas[]

Correspond to the same syntax elements of the same name in the reference software about loop filter deltas applied at the macroblock (MB) level based on mode or reference frame, and affect the decoding processes of the deblocking filter accordingly.

use_prev_in_find_mv_refs

Indicates whether the previous mode information context from the last decoded frame can be used or not.

ReservedControlInfo5Bits

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

wControlInfoFlags

Provides an alternative way to access the bit fields.

base_qindex, y_dc_delta_q, uv_dc_delta_q, uv_ac_delta_q

Correspond to the same syntax elements of the same name in the reference software about quantization parameters applied for the Y, U and V planes, and affect the decoding processes of inverse quantization accordingly.

stVP9Segments

Provides the segmentation related syntax values when segmentation is enabled, including the control flags of enabled, abs_delta, and segmentation map and data related tables **tree_probs[]**, **pred_probs[]**, **feature_data[][]**, and **feature_mask[]**.

> **enabled** in **stVP9Segments**
>
> Indicates whether segmentation map related syntax elements are present or not for current frame. If enabled is equal to 0, it indicates that segmentation map related syntax elements are not present for the current frame and the control flags of abs_delta and segmentation map related tables **tree_probs[]**, **pred_probs[]**, **feature_data[][]**, and **feature_mask[]** are not valid and shall be ignored by accelerator.
>
> **update_map** and **temporal_update** in **stVP9Segments**
>
> Indicate whether **tree_probs[]** and **pred_probs[]** are to be updated by syntax elements in the frame or not.
>
> **tree_probs[]** and **pred_probs[]** in **stVP9Segments**
>
> Provide segmentation map for current frame and affect the decoding process accordingly.
>
> **abs_delta, feature_data[][],**and **feature_mask[]** in **stVP9Segments**
>
> Indicate segmentation data for current frame and affect the decoding process accordingly.
>
> **ReservedSegmentFlags4Bits**
>
> Reserved bit fields. Shall be set to 0. The accelerator shall ignore the values in the reserved bit fields.
>
> **wSegmentInfoFlags**
>
> Provides an alternative way to access the bit fields.

log2_tile_cols, log2_tile_rows

Correspond to the same syntax elements of the same name in the reference software about tile partitions, and affect the decoding process of tiles accordingly.

VP9 supports tiles, where the picture is broken up into a grid of tiles along superblock boundaries. Superblocks have the size of 64x64 in luma sample units. The tiles are always as evenly spaced as possible, and there are a power-of-two number of them. Tiles must be at least 256 luma samples wide and must be no more than 4096 luma samples wide. There can be no more than four tile rows. The tiles are scanned in raster scan order, and the super blocks within them are coded in raster scan order within each tile. Thus the ordering of superblocks within the frame depends on the tile structure. Coding dependencies are broken along vertical tile boundaries, which means that two tiles in the same tile row may be decoded at the same time. Coding dependencies are not broken between horizontal boundaries. Thus, for example, a frame split into 2x2 tiles can be decoded with two cores/threads operating in parallel, but not with four.

At the start of every tile except the last one, a 32-bit byte count is transmitted, indicating how many bytes are used to code the next tile. This lets a decoder with parallel computing capability skip ahead to the next tile in order to start a parallel decoding task.

uncompressed_header_size_byte_aligned

Corresponds to the size of uncompressed header in bytes (with byte alignment). The accelerator may choose to skip the parsing of the uncompressed header and start parsing of the compressed header and block data using the parameter of uncompressed_header_size_byte_aligned.

first_partition_size

Corresponds to the size of the compressed header data partition in bytes. The bitstream data buffer shall contain the uncompressed header in the size of **uncompressed_header_size_byte_aligned**, the compressed header with the size equal to **first_partition_size,** and the compressed frame data. The accelerator can identify the beginning of the compressed block-level data by using the sum of the offsets, **uncompressed_header_size_byte_aligned** and **first_partition_size** which corresponds to the uncompressed header size and the compressed header size in bytes.

Reserved16Bits, Reserved32Bits

Reserved bit fields. Shall be set to 0 by the host decoder and the accelerator shall ignore their value.

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.

4. VP8 Picture Parameters Data Structure

The **DXVA_PicParams_VP8** structure provides the picture-level parameters of a compressed picture for VP8 video decoding. This structure is used when bDXVA_Func is 1 and the buffer type is DXVA2_PictureParametersBufferType (in DXVA 2.0).

4.1 Syntax

```
typedef struct _segmentation_VP8 {
      union {
        struct {
            UCHAR      segmentation_enabled             : 1;
            UCHAR      update_mb_segmentation_map       : 1;
            UCHAR      update_mb_segmentation_data      : 1;
            UCHAR      mb_segement_abs_delta            : 1;
            UCHAR      ReservedSegmentFlags4Bits        : 4;
        };
        UCHAR    wSegmentFlags;
      };
      CHAR  segment_feature_data[2][4];
      UCHAR mb_segment_tree_probs [3];
} DXVA_segmentation_VP8;


typedef struct _DXVA_PicParams_VP8 {
      UINT       first_part_size;
      UINT       width;
      UINT       height;
      DXVA_PicEntry_VPx     CurrPic;
      union {
        struct {
            UCHAR      frame_type                       : 1;
            UCHAR      version                          : 3;
            UCHAR      show_frame                       : 1;
            UCHAR      clamp_type                       : 1;
            UCHAR      ReservedFrameTag3Bits            : 2;
        };
        UCHAR    wFrameTagFlags;
      };
      DXVA_segmentation_VP8  stVP8Segments;
```

```
  UCHAR       filter_type;
  UCHAR       filter_level;
  UCHAR       sharpness_level;
  UCHAR       mode_ref_lf_delta_enabled;
  UCHAR       mode_ref_lf_delta_update;
  CHAR        ref_lf_deltas[4];
  CHAR        mode_lf_deltas[4];
  UCHAR       log2_nbr_of_dct_partitions;
  UCHAR       base_qindex;
  CHAR        y1dc_delta_q;
  CHAR        y2dc_delta_q;
  CHAR        y2ac_delta_q;
  CHAR        uvdc_delta_q;
  CHAR        uvac_delta_q;
  DXVA_PicEntry_VPx     alt_fb_idx;
  DXVA_PicEntry_VPx     gld_fb_idx;
  DXVA_PicEntry_VPx     lst_fb_idx;
  UCHAR       ref_frame_sign_bias_golden;
  UCHAR       ref_frame_sign_bias_altref;
  UCHAR       refresh_entropy_probs;
  UCHAR       vp8_coef_update_probs[4][8][3][11];
  UCHAR       mb_no_coeff_skip;
  UCHAR       prob_skip_false;
  UCHAR       prob_intra;
  UCHAR       prob_last;
  UCHAR       prob_golden;
  UCHAR       intra_16x16_prob[4];
  UCHAR       intra_chroma_prob[3];
  UCHAR       vp8_mv_update_probs[2][19];
  USHORT      ReservedBits1;
  USHORT      ReservedBits2;
  USHORT      ReservedBits3;
  UINT        StatusReportFeedbackNumber;
} DXVA_PicParams_VP8, *LPDXVA_PicParams_VP8;
```

4.2 Semantics

first_part_size

Determines the size of the first several partitions (control partitions), including the first compressed (bool coded) partition for header information that applies to the frame as a whole, the second compressed (bool coded) partition for per-macroblock information specifying how each macroblock is predicted from the already-reconstructed data that is available to the decoding process. The accelerator may choose to skip the parsing of the uncompressed header (10 bytes for a key frame or 3 bytes for a non-key frame) and the partitions indicated by **first_part_size** and start decoding block data. The bitstream data buffer shall contain the uncompressed header (3 bytes for inter frames and 10 bytes for key frames), the first compressed (bool coded) partition for header information that applies to the frame as a whole, the second compressed (bool coded) partition for per-macroblock information specifying how each macroblock is predicted from the already-reconstructed data that is available to the decoding process, and other partitions for the DCT/WHT coefficients (quantized and logically compressed) of the residue signal to be added to the predicted block values in each block.

width, height

Specify the coded width and height of current frame. Correspond to the syntax elements of the same name in the reference software and affect the decoding process accordingly. Coded resolution changes are only allowed to occur on key frames in VP8. Under the DXVA framework, the optional upscaling of decoded pictures prior to display is performed outside of the VP8 accelerator, when horiz_scale and/or vert_scale are non-zero.

CurrPic

Specifies the destination frame buffer/surface index for the decoded picture. In this context, the **AssociatedFlag** has no meaning and shall be 0, and the accelerator shall ignore its value.

frame_type

Indicates the current frame type. Allowed values are 0 (KEY_FRAME) and 1 (INTER_FRAME).

version

Version number that enables or disables certain features in the bitstream, as specified in the Internet-Draft of the VP8 bitstream specification.

show_frame

Indicates whether the current frame is meant to be displayed or not. The accelerator might use the flag for internal optimizations.

clamp_type

Specifies whether the decoder is required to clamp the reconstructed sample values. Allowed values are 1 (RECON_CLAMP_REQUIRED) and 1 (RECON_CLAMP_NOTREQUIRED), as defined in the reference software.

ReservedFrameTag3Bits

Reserved bit fields. Shall be set to 0 by the host decoder and the accelerator shall ignore their value.

wFrameTagFlags

Provides an alternative way to access the bit fields.

stVP8Segments

Provides the segmentation related syntax values when segmentation is enabled, including the control flags of **update_mb_segmentation_map**, **update_mb_segmentation_data**, **mb_segement_abs_delta** and segmentation map and data related tables **segment_feature_data[][]**,and **mb_segment_tree_probs[]**.

segmentation_enabled in **stVP8Segments**

Enables the segmentation feature for the current frame. When **segmentation_enabled** is equal to 0, the accelerator shall ignore the values in **stVP8Segments.** VP8 uses segment based adjustments to support changing the quantizer level and loop filter level for a macroblock. When the segment-based adjustment feature is enabled for a frame, each macroblock within the frame is coded with a segment_id. This results in segmenting the macroblocks of the current frame into a number of different segments. Macroblocks within the same segment use the same for quantizer and loop filter level adjustments.

update_mb_segmentation_map in **stVP8Segments**

Determines whether the MB segmentation map is updated in the current frame.

update_segment_feature_data in **stVP8Segments**

Indicates whether the segment feature data is updated in the current frame.

mb_segement_abs_delta in **stVP8Segments**

Indicates the feature data update mode, 0 for delta and 1 for the absolute value.

segment_feature_data[][] in **stVP8Segments**

Indicates the alternate quantizer and alternate loop filter value for segments.

mb_segment_tree_probs [] in **stVP8Segments**

Indicates the branch probabilities of the segment_id decoding tree.

ReservedSegmentFlags4Bits

Reserved bit fields. Shall be set to 0 by the host decoder and the accelerator shall ignore their value.

wSegmentFlags

Provides an alternative way to access the bit fields.

filter_type, filter_level, and **sharpness_level**

filter_type determines whether the normal or the simple loop filter is used, **filter_level** controls the deblocking filter, and **sharpness_level** controls the deblocking filter.

mode_ref_lf_delta_enabled, mode_ref_lf_delta_update

Indicate whether the MB-level loop filter adjustment (based on the used reference frame and coding mode) is on for the current frame, and whether the delta values used in adjustment are updated in the current frame.

ref_lf_deltas[]

Specify the adjustment delta values corresponding to a certain used reference frame for loop filtering.

mode_lf_deltas[]

Specify the adjustment delta values corresponding to certain MB prediction mode for loop filtering.

log2_nbr_of_dct_partitions

Determines the number of separate partitions containing the DCT coefficients of the macroblocks.

base_qindex

Specifies the dequantization table index used for the luma AC coefficients (and other coefficient groups if no delta value is present).

y1dc_delta_q, y2dc_delta_q, y2ac_delta_q, uvdc_delta_q, uvac_delta_q

Indicate the delta values that are added to the baseline index to obtain the luma DC coefficient dequantization index, the Y2 block DC coefficient dequantization index, the Y2 block AC coefficient dequantization index, the chroma DC coefficient dequantization index, and the chroma AC coefficient dequantization index.

alt_fb_idx, gld_fb_idx, lst_fb_idx

Specify the frame buffer/surface indices for the altref frame, the golden frame, and the previous reconstructed frame. In this context, the AssociatedFlag has no meaning and shall be 0, and the accelerator shall ignore its value. The host decoder may set **alt_fb_idx, gld_fb_idx,** and **lst_fb_idx** parameters according to the bitstream syntax elements of copy_buffer_to_arf, copy_buffer_to_gf, refresh_golden_frame, refresh_alt_ref_frame, refresh_last_frame. The host decoder may set the **alt_fb_idx, gld_fb_idx,** and **lst_fb_idx** parameters different from the indication of those bitstream syntax elements on purpose. The accelerator shall honor the host decoder settings of **alt_fb_idx, gld_fb_idx,** and **lst_fb_idx** parameters.

The VP8 decoder needs to maintain four YUV frame buffers/surfaces for decoding purposes. These buffers hold the current frame being reconstructed, the previous reconstructed frame, the most recent golden frame, and the most recent altref frame.

ref_frame_sign_bias_golden, ref_frame_sign_bias_altref

ref_frame_sign_bias_golden controls the sign of motion vectors when the golden frame is referenced and **ref_frame_sign_bias_altref** controls the sign of motion vectors when the altref frame is referenced.

refresh_entropy_probs

Determines whether updated token probabilities are used only for this frame or until further update.

vp8_coef_update_probs[][][][]

Indicate the new branch probability for different block types, coefficient bands, coefficient contexts, and entropy coding nodes. The forward probability updates are accumulative. That is, a probability updated on one frame is in effect for all subsequent frames until the next key frame or until the probability is explicitly updated by another frame.

mb_no_coeff_skip

Enables or disables the skipping of macroblocks containing no non-zero coefficients.

prob_skip_false

Indicates the probability estimate that the macroblock is not skipped (flag indicating skipped macroblock is false).

prob_intra, prob_last, prob_golden

Indicate the probability estimate of an intra macroblock, the probability estimate that the last reference frame is used for inter prediction, and the probability estimate that the golden reference frame is used for inter prediction.

intra_16x16_prob[]

Specify the branch probability estimates of the luma intra prediction mode decoding tree.

intra_chroma_prob[]

Specify the branch probability estimates of the chroma intra prediction mode decoding tree.

vp8_mv_update_probs[][]

Specify the corresponding MV decoding probability estimates for the current frame.

ReservedBits1, ReservedBits2, ReservedBits3

Reserved bit fields. Shall be set to 0 by the host decoder and the accelerator shall ignore their value.

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. Slice (Picture) Control Data Structure

The slice control structure is used when bDXVA_Func is 1 and the buffer type is DXVA2_SliceControlBufferType (DXVA 2.0). The slice control buffer is accompanied by a raw bitstream data buffer. 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. Since both VP8 and VP9 do not have multiple slice support, each picture contains one slice and only one slice control buffer shall be present for a frame.

Only the **DXVA_Slice_VPx_Short** structure is defined in this specification.

5.1 Syntax

The **DXVA_Slice_VPx_Short** data structure, as specified for other DXVA usage cases, is sent by the host software decoder to the accelerator to convey slice control data. The data structure and associated semantics are essentially the same as for the previous DXVA_Slice_H264_Short and DXVA_Slice_HEVC_Short data structures. It has been given a new name so that the data structures used for VP8 and VP9 will have names that are associated with the new design.

For convenience, the form of this data structure is shown below:

```
typedef struct _DXVA_Slice_VPx_Short {
      UINT        BSNALunitDataLocation;
      UINT        SliceBytesInBuffer;
      USHORT      wBadSliceChopping;
} DXVA_Slice_VPx_Short, *LPDXVA_Slice_VPx_Short;
```

5.2 Semantics

BSNALunitDataLocation

If **wBadSliceChopping** is 0 or 1, this member locates the compressed bitstream data for the current frame. The value is the byte offset, from the start of the bitstream data buffer, of the first byte of the compressed frame.

If **wBadSliceChopping** is not 0 or 1, **BSNALunitDataLocation** shall be 0.

SliceBytesInBuffer

Number of bytes in the bitstream data buffer that are associated with this slice control data structure, starting with the byte at the offset given in **BSNALunitDataLocation**.

wBadSliceChopping

Contains one of the following values:

Value	Description
0	All bits for the slice are located within the corresponding bitstream data buffer.
1	The bitstream data buffer contains the start of the slice, but not the entire slice, because the buffer is full.
2	The bitstream data buffer contains the end of the slice. It does not contain the start of the slice, because the start of the slice was located in the previous bitstream data buffer.
3	The bitstream data buffer does not contain the start of the slice (because the start of the slice was located in the previous bitstream data buffer), and it does not contain the end of the slice (because the current bitstream data buffer is full).

Generally the host decoder should avoid using values other than 0 for **wBadSliceChopping**.

The size of the data in the bitstream data buffer (and the amount of data reported by the host software decoder) shall be an integer multiple of 128 bytes. When **wBadSliceChopping** is 0 or 2, if the end of the slice data is not an even multiple of 128 bytes, the decoder should pad the end of the buffer with zeroes.

The host decoder is recommended to send only decodable compressed frames to accelerator. However, the accelerator shall be robust enough to handle any non-decodable frames.

6. Status Report Data Structure

The **DXVA_Status_VPx** 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_VPx** 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_VPx** 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 and DXVA_Status_HEVC data structures. It has been given a new name so that the data structures used for VP8 and VP9 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_VPx {
  UINT  StatusReportFeedbackNumber;
  DXVA_PicEntry_VPx CurrPic;
  UCHAR  bBufType;
  UCHAR  bStatus;
  UCHAR  bReserved8Bits;
  USHORT wNumMbsAffected;
} DXVA_Status_VPx, *LPDXVA_Status_VPx;
```

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 VP8 and VP9 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_ModeVP9_VLD_Profile0 Profile

This profile supports the features necessary for a decoder that conforms to VP9 profile 0 (which supports 8 bit 4:2:0 video). 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.

All data buffers shall contain only data that is consistent with the constraints specified for VP9 profile 0 (including the constraint that **BitDepthMinus8Luma** and **BitDepthMinus8Chroma** shall both be equal to 0).

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

```
// {463707F8-A1D0-4585-876D-83AA6D60B89E}

DEFINE_GUID(DXVA_ModeVP9_VLD_Profile0,

0x463707f8, 0xa1d0, 0x4585, 0x87, 0x6d, 0x83, 0xaa, 0x6d, 0x60, 0xb8,
0x9e);
```

7.2 DXVA_ModeVP9_VLD_10bit_Profile2 Profile

This profile supports the features necessary for a decoder that conforms to VP9 profile 2 for 10 bit video. 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.

All data buffers shall contain only data that is consistent with the constraints specified for VP9 profile 2 for 10 bit video (including the constraint that **BitDepthMinus8Luma** and **BitDepthMinus8Chroma** shall both be equal to 2).

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

```
// {A4C749EF-6ECF-48AA-8448-50A7A1165FF7}

DEFINE_GUID(DXVA_ModeVP9_VLD_10bit_Profile2,

0xa4c749ef, 0x6ecf, 0x48aa, 0x84, 0x48, 0x50, 0xa7, 0xa1, 0x16, 0x5f,
0xf7);
```

7.3 DXVA_ModeVP8_VLD Profile

This profile supports the features necessary for a decoder that conforms to the VP8 coding format. 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.

All data buffers shall contain only data that is consistent with the constraints specified for the VP8 coding format.

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

```
// {90B899EA-3A62-4705-88B3-8DF04B2744E7}

DEFINE_GUID(DXVA_ModeVP8_VLD,

0x90b899ea, 0x3a62, 0x4705, 0x88, 0xb3, 0x8d, 0xf0, 0x4b, 0x27, 0x44,
0xe7);
```

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.