AMD Accelerated Parallel Processing TECHNOLOGY

OpenVideo Decode (OVD) API

1 Overview

This document specifies the OpenVideo Decode (OVD) API definitions. This API is designed to provide platform-independent video codec functions using hardware accelerators.

OpenVideo Decode API has the following design goals and highlights:

- OpenVideo Decode API is defined for bitstream based video decoding.
- This version of the Open Video Decode API supports H.264 decode.
- OpenVideo Decode API is extendable to support other standard video codecs.
- OpenVideo API may be extended to support hardware accelerated encoding functions in the future.
- The number of accelerated multiple video streams is not limited by the API but may be limited by the hardware accelerator.

OpenVideo supports the GPU fixed-function hardware Unified Video Decoder (UVD), which allows interoperability with OpenCL through a common API (OpenDecode API). OpenVideo provides the way for all OpenCL-based video applications to access the fixed-function hardware in GPUs. The video application can use OpenVideo as part of its video pipeline for video playback, video editing, or video transcoding.

The OpenDecode API is the part of OpenVideo that allows applications to use the GPU's UVD engine. OpenVideo supports full bitstream decoding acceleration. The decoded output then can be used for either displaying directly using the GPU, or for other post-processing operations through Open CL kernels run on the GPU shaders (post-process filtering or transcoding operations). The OpenVideo API is fully interoperable with the OpenCL API: it allows for shared surfaces between the two domains.

Figure 1 illustrates the various pipeline possibilities for transcoding video.

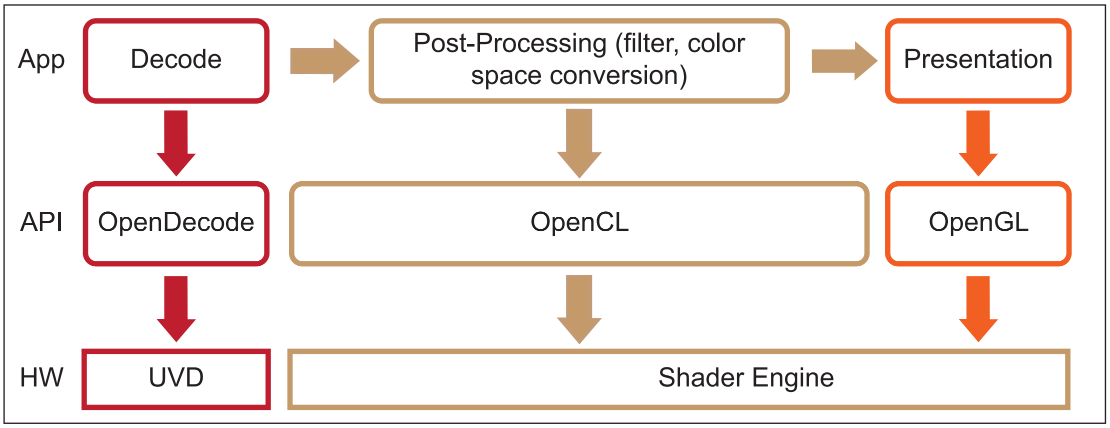

Figure 1 Video Pipeline Possibilities with Multiple GPUs and Multiple APIs

The developer can use the OpenVideo APIs to access fixed-function codec engines in OpenCL.

2 Interoperability of OpenCL and OpenVideo

The OVD API operates with other domains, such as OpenCL. The domain API provides functions for application to obtain the platform-independent handle that can be used by another domain API. AMD OpenCL provides extension functions to obtain the platform independent handle that can be used by OpenVideo. The OpenCL extension functions used to obtain the platform-independent handle for context, memory object, and event object are briefly described in the following subsections.

2.1 Platform Context

```
OPContextHandle clGetPlatformContextHandle(cl_context OpenCLContextHandle);
```

`clGetPlatformContextHandle` retrieves a domain-independent context handle from an OpenCL context.

2.2 Platform Memory Object

```
OPMemHandle clGetPlatformMemHandle(cl_mem OpenCLMemoryHandle);
```

`clGetPlatformMemHandle` retrieves a domain-independent memory handle from an OpenCL memory object.

2.3 Platform Event Object

```
OPEventHandle clGetPlatformEventHandle(cl_event OpenCLEventHandle);
```

`clGetPlatformEventHandle` retrieves a domain-independent handle from an OpenCL event object.

Platform handles can be used by OpenVideo platform functions to obtain the OpenVideo objects.

This section describes the high-level operation sequence in an application by using OVD and OpenCL APIs. The flow diagram shown in Figure 2 shows a simple application using OpenDecode API to decode the video stream and OpenCL to render the decoded video.

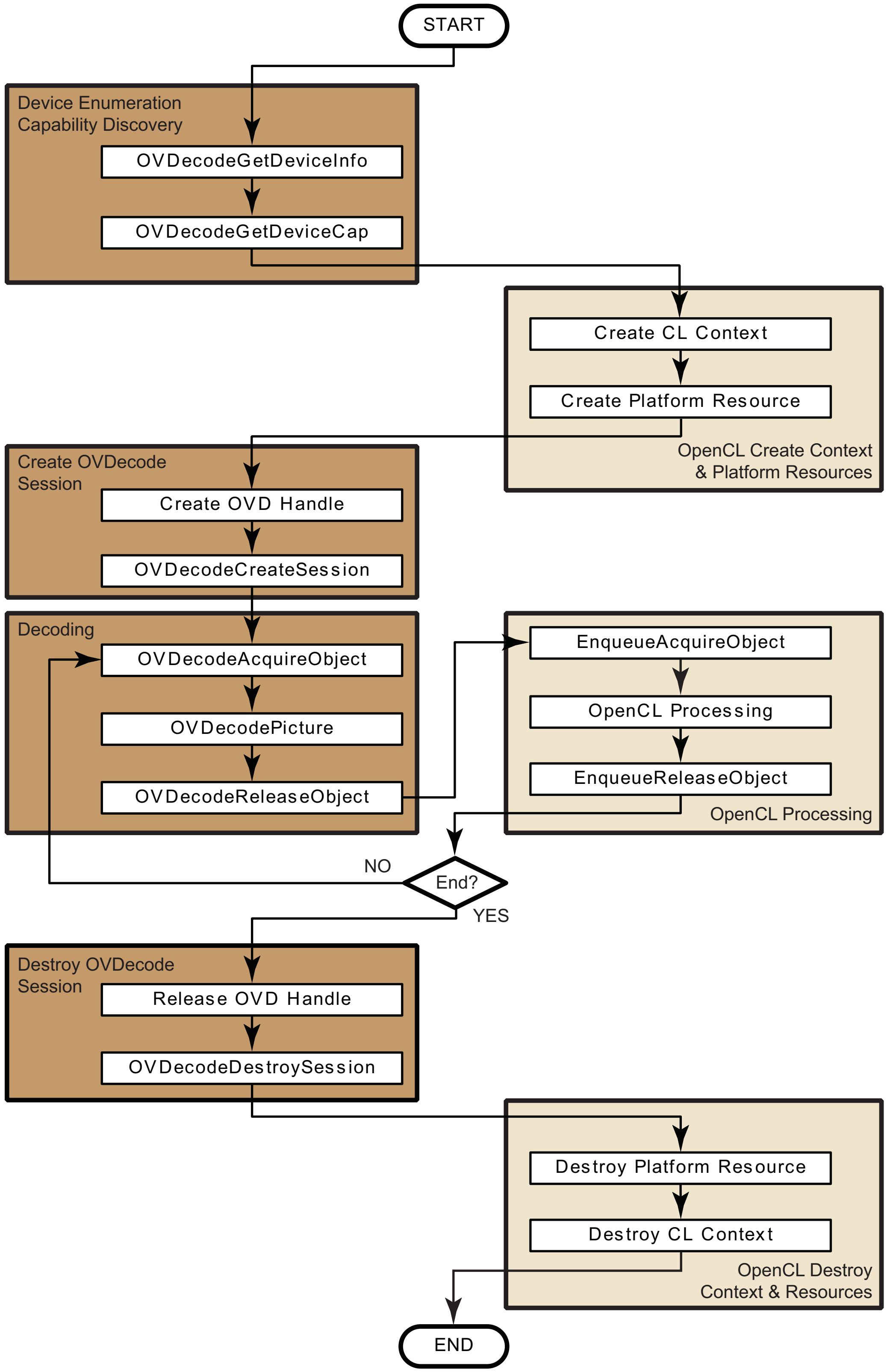

Figure 2 Operational Flow of OpenVideo Decode API with OpenCL Processing

3 OpenVideo Decode API Functions

The functions described in this section are grouped by use. Table 1 lists these functions in alphabetic order.

Table 1 Function Listing

OVDecodeGetDeviceInfo

Prerequisite

```
typedef struct {
  unsigned int      device_id;
  unsigned int      max_decode_stream;
  unsigned int      decode_cap_size;
} ovdecode_device_info;
```

Description

Queries the available decode device. The ovdecode_device_info contains a unique device_id and the size of the decode_cap structure for each available device. The decode_cap_size specifies the data structure size of decode_cap in OVDecodeGetDeviceCap that the application must provide when it calls this function.

Return

```
OVresult OVDecodeGetDeviceInfo(
    unsigned int          *num_device,
    ovdecode_device_info  *device_info
);
```

OVresult

1 = available.
0 = unavailable.

OVDecodeGetDeviceCap

Prerequisite

```
// OpenVideo Decode Profile
typedef enum
{
   OVD_H264_BASELINE_41 = 1,// H.264 bitstream acceleration baseline profile up to level 4.1
   OVD_H264_MAIN_41,        // H.264 bitstream acceleration main profile up to level 4.1
   OVD_H264_HIGH_41,        // H.264 bitstream acceleration high profile up to level 4.1
   OVD_H264_BASELINE_51,    // H.264 bitstream acceleration baseline profile up to level 5.1
   OVD_H264_MAIN_51,        // H.264 bitstream acceleration main profile up to level 5.1
   OVD_H264_HIGH_51,        // H.264 bitstream acceleration high profile up to level 5.1
   OVD_H264_STEREO_HIGH     // H.264 bitstream acceleration stereo high profile
   OVD_VC1_SIMPLE,          // VC-1 bitstream acceleration simple profile
   OVD_VC1_MAIN,            // VC-1 bitstream acceleration main profile
   OVD_VC1_ADVANCED,        // VC-1 bitstream acceleration advanced profile
} ovdecode_profile;


// OpenVideo Decode Format
typedef enum
{
  OVD_NV12_INTERLEAVED = 1,// NV12 Linear Interleaved
} ovdecode_format;


typedef struct {
  ovdecode_profile profile;          // codec information about the decode capability
  ovdecode_format  output_format;    // decode output format supported in this device
} ovdecode_cap;
```

Description

Queries the decoder capability, including codec information and output format, that the device can support.The decoder capability is obtained from the specified device_id.

Return

```
OVresult OVDecodeGetDeviceCap (
    unsigned int          device_id,
    unsigned int          num_of_decode_cap,
    ovdecode_cap          *decode_cap_list
);
```

OVresult

1 = available.
0 = unavailable.

OVCreateOVDHandleFromOPHandle

Prerequisite

```
#define OPMemHandle void *
```

Description

Creates the decode handle from the platform memory handle. The decode handle can be used in the OVDecodePicture function as the output decode buffer. The application can create multiple output buffers to queue the decode job.

Return

```
ov_handle OVCreateOVDHandleFromOPHandle (
    OPMemHandle           platform_memhandle
);
```

OVresult

1 = available.
0 = unavailable.

OVReleaseOVDHandle

Prerequisite

```
#define ov_handle HANDLE
```

Description

Releases the decode handle. After release, the handle is invalid and cannot be used for decode picture.

Return

```
OVresult OVReleaseOVDHandle (
    ov_handle              decode_handle
);
```

OVresult

1 = success.
0 = fail.

OVAcquireObject

Prerequisite

```
#define ov_session void *

#define ovd_event void *
```

Description

Acquires the memory objects that have been created from OpenCL. These objects must be acquired before they can be used by the decode function.

Return

```
OVresult OVAcquireObject (
    ov_session             session,
    unsigned int           num_handle,
    ov_handle              *decode_handle,
    unsigned int           num_event_in_wait_list,
    ovd_event              *event_wait_list,
    ovd_event              *event
);
```

OVresult

1 = available.
0 = unavailable.

OVReleaseObject

Prerequisite

```
#define ov_session void *

#define ovd_event void *
```

Description

Releases the memory objects created from OpenCL. The objects must be released before they can be used by OpenCL.

Return

```
OVresult OVReleaseObject (
    ov_session             session,
    unsigned int           num_handle,
    ov_handle              *decode_handle,
    unsigned int           num_event_in_wait_list,
    ovd_event              *event_wait_list,
    ovd_event              *event
);
```

OVresult

1 = success.
0 = fail.

OVDecodeCreateOVDEventFromOPEventHandle

Prerequisite	#define OPEventHandle HANDLE
Description	Creates the OVD event handle from the platform event handle.

Return	`OVresult OVDecodeCreateOVDEventFromOPEventHandle (`
	`    OPEventHandle          platform_eventhandle`
	`);`

OVresult	1 = available. 0 = unavailable.

OVDecodeReleaseOVDEventHandle

Prerequisite	#define ovd_event void *
Description	Releases the OVD event handle.

Return	`OVresult OVDecodeReleaseOVDEventFromOPEventHandle (`
	`    ovd_event              ovd_event`
	`);`

OVresult	1 = success. 0 = fail.

OVDecodeCreateSession

Prerequisite

```
#define OPContext void *

// OpenVideo Decode Profile
typedef enum
{
OVD_H264_BASELINE_41 = 1,// H.264 bitstream acceleration baseline profile up to level 4.1
OVD_H264_MAIN_41, // H.264 bitstream acceleration main profile up to level 4.1
OVD_H264_HIGH_41, // H.264 bitstream acceleration high profile up to level 4.1
OVD_H264_BASELINE_51, // H.264 bitstream acceleration baseline profile up to level 5.1
OVD_H264_MAIN_51, // H.264 bitstream acceleration main profile up to level 5.1
OVD_H264_HIGH_51, // H.264 bitstream acceleration high profile up to level 5.1
OVD_H264_STEREO_HIGH // H.264 bitstream acceleration stereo high profile
OVD_VC1_SIMPLE, // VC-1 bitstream acceleration simple profile
OVD_VC1_MAIN, // VC-1 bitstream acceleration main profile
OVD_VC1_ADVANCED, // VC-1 bitstream acceleration advanced profile
} ovdecode_profile;

// OpenVideo Decode Format
typedef enum
{
OVD_NV12_INTERLEAVED = 1,// NV12 Linear Interleaved
} ovdecode_format;
```

Description

Creates the decode session for each decoding stream. After the session creation, the decoder can accept the decode picture job from the application. For decoding multiple streams, the application can create multiple sessions within the same platform context; the application is responsible for managing the input and output buffers for each decode session.

Return

```
ov_session OVDecodeCreateSession (
    OPContext            platform_context,
    unsigned int         device_id,
    ovdecode_profile     profile,
    ovdecode_format      output_format,
    unsigned int         output_width,
    unsigned int         output_height
);
```

OVresult

1 = ov_session available.
0 = unavailable.

OVDecodePicture

Prerequisite

```
typedef struct
{
unsigned int codec_type;
unsigned int profile;
unsigned int level;
unsigned int width_in_mb;
unsigned int height_in_mb;
unsigned int decode_flag; // Reserved for future features - always 0
void *reserved_reference [16]; // Reserved - Not used for bitstream decoding
unsigned int reserved [15]; // Reserved for future features - always 0
} ovd_picture_parameter_1;

#define ovd_bitstream_data unsigned char *

typedef struct
{
unsigned int SliceBitsInBuffer;
unsigned int SliceDataLocation;
unsigned int SliceBytesInBuffer;
unsigned int reserved[5];
} ovd_slice_data_ctrl;

#define ov_session void *

#define ov_handle HANDLE
```

Description

Decodes a single picture. For decoding multiple streams, the decode picture jobs from different streams can be interleaved in any order.

Return

```
OVresult OVDecodePicture (
    ov_session                  session,
    ovd_picture_parameter       *picture_parameter_1,
    void                        *picture_parameter_2,
    unsigned int                picture_parameter_2_size,
    ovd_bitstream_data          *bitstream_data,
    unsigned int                bitstream_data_size,
    ovd_slice_data_control      *slice_data_control,
    unsigned int                slice_data_control_size,
    ov_handle                   output_handle,
    unsigned int                num_event_in_wait_list,
    ovd_event                   *event_wait_list,
    ovd_event                   *event,
    unsigned int                picture_id
);
```

OVresult

1 = success.
0 = fail.

OVDecodeDestroySession

Prerequisite	`#define ov_session void *`
Description	Destroys the decode session. Destroying a session releases all associated hardware resources, and no further decoding work can be performed with the session.
Return	`OVresult OVDecodeDestroySession ( ov_session session );`
OVresult	1 = success. 0 = fail.

4 Decode Data Buffers

`OVDecodePicture` requires four decode buffer types for bitstream decoding:

- `OVD_PICTURE_PARAMETER_1`
- `PICTURE_PARAMETER_2`
- `OVD_BITSTREAM_DATA`
- `OVD_SLICE_CONTROL_DATA`

These buffer types are described in the following subsections.

NOTE: The compressed data buffers have a different data/bit layout for H.264, VC-1, and MPEG2. Read the spec carefully and implement the appropriate buffer for each codec

4.1 OVD_PICTURE_PARAMETER_1

This buffer contains the common information of the current decoded picture for all supported codecs. This structure definition is defined as:

```
typedef struct
{
     unsigned int          codec_type;
     unsigned int          profile;
     unsigned int          level;
     unsigned int          width_in_mb;
     unsigned int          height_in_mb;

     unsigned int          decode_flag;                 // Reserved for future features - always 0
     void                  *reserved_reference [16];    // Reserved - Not used for bitstream decoding
     unsigned int          reserved [15];               // Reserved for future features - always 0

} ovd_picture_parameter_1;
```

The description of each field in the data structure is defined as follows:

Table 2   Data Structure Fields

Parameter	Description
codec_type	Specifies the codec type of the current decode picture 1 = H.264 2 = VC-1
profile	Specifies a subset of algorithmic features and limits that must be supported by all decoders conforming to that profile. All H.264 profile types are specified in the H.264 specification. All VC-1 profile types are specified in the VC-1 specification. AMD UVD hardware acceleration supports the following profiles. H.264: 1 = Baseline profile (for H264 field *profile_idc* = 66) 2 = Main profile (for H264 field *profile_idc* = 77) 3 = High profile (for H264 field *profile_idc* = 100) VC-1: 4 = Simple profile (for the VC-1 field *PROFILE* = 0) 5 = Main profile (for the VC-1 field *PROFILE* = 1) 6 = Advanced profile (for the VC-1 field *PROFILE* = 3)
level	Specifies restrictions on bitstreams, as well as limits on the capabilities needed to decode the bitstreams. Levels are specified within each profile.
width_in_mb	Specifies the width of each decoded picture in units of macroblocks.
height_in_mb	Specifies the height of each decoded picture in units of macroblocks.
decode_flag	Reserved for future usage, always 0.
reserved_reference [16]	Not used, always 0.
reserved [15]	Not used, always 0.

4.2 PICTURE_PARAMETER_2

This buffer contains the codec-specific information of the current decoded picture. The application must pass in the corresponding `picture_parameter_2` structure in the OVDecodePicture call based on the codec type of the decode picture. The `picture_parameter_2` structures for each codec are defined as described in the following subsections.

4.2.1 H.264 picture_parameter_2 Structure

```
typedef struct
{
   union{
      struct {
         unsigned int    residual_colour_transform_flag      : 1;
         unsigned int    delta_pic_always_zero_flag          : 1;
         unsigned int    gaps_in_frame_num_value_allowed_flag : 1;
         unsigned int    frame_mbs_only_flag                 : 1;
         unsigned int    mb_adaptive_frame_field_flag        : 1;
         unsigned int    direct_8x8_inference_flag           : 1;
         unsigned int    sps_reserved                        : 26;
      } sps_flag;
      unsigned intflag;
   }sps_info;

   union {
      struct {
         unsigned int    entropy_coding_mode_flag              : 1;
         unsigned int    pic_order_present_flag                : 1;
         unsigned int    weighted_pred_flag                    : 1;
         unsigned int    weighted_bipred_idc                   : 2;
         unsigned int    deblocking_filter_control_present_flag : 1;
         unsigned int    constrained_intra_pred_flag           : 1;
         unsigned int    redundant_pic_cnt_present_flag        : 1;
         unsigned int    transform_8x8_mode_flag               : 1;
         unsigned int    pps_reserved                          : 23;
      } pps_flag;
      unsigned intflag;
   pps_info;

   unsigned int      picture_structure;
   unsigned char     chroma_format;
   unsigned char     bit_depth_luma_minus8;
   unsigned char     bit_depth_chroma_minus8;
   unsigned char     log2_max_frame_num_minus4;

   unsigned char     pic_order_cnt_type;
   unsigned char     log2_max_pic_order_cnt_lsb_minus4;
   unsigned char     num_ref_frames;
   unsigned char     reserved_8bit;

   char              pic_init_qp_minus26;
   char              pic_init_qs_minus26;
   char              chroma_qp_index_offset;
   char              second_chroma_qp_index_offset;

   unsigned char     num_slice_groups_minus1;
   unsigned char     slice_group_map_type;
   unsigned char     num_ref_idx_l0_active_minus1;
   unsigned char     num_ref_idx_l1_active_minus1;

   unsigned short    slice_group_change_rate_minus1;
   unsigned short    reserved_16bit;

   unsigned char     scaling_lists_4x4[6][16];
   unsigned char     scaling_lists_8x8[2][64];

   unsigned int      frame_num;
   unsigned int      frame_num_list[16];// bit 31 is used to indicate long/short term
   int               curr_field_order_cnt_list[2];
   int               field_order_cnt_list[16][2];

   int               intra_flag;
```

```
    struct {
       unsigned int   numViews;
       unsigned int   viewID0;
       mvcElement_t   mvcElements [1];          // Allocate numViews-1 elements here
    } mvc;

    unsigned int      reserved[128];

} H264_picture_parameter_2;

typedef struct
{
    unsigned short    viewOrderIndex;
    unsigned short    viewID;
    unsigned short    numOfAnchorRefsInL0;
    unsigned short    viewIDofAnchorRefsInL0[15];
    unsigned short    numOfAnchorRefsInL1;
    unsigned short    viewIDofAnchorRefsInL1[15];
    unsigned short    numOfNonAnchorRefsInL0;
    unsigned short    viewIDofNonAnchorRefsInL0[15];
    unsigned short    numOfNonAnchorRefsInL1;
    unsigned short    viewIDofNonAnchorRefsInL1[15];
} mvcElement_t;
```

Table 3 through Table 6 describe each field in the H264_picture_parameter_2 data structure. Unless indicated otherwise, the parameters in Table 3 correspond to the same-named fields in the H.264/ACV1 specification.

Table 3 sps_info Structure Flags

Parameter	Description
residual_colour_transform_flag	1 = residual color transform is applied. 0 = residual color transform is not applied. When residual_colour_transform_flag is not present, it is assumed to be 0.
delta_pic_order_always_zero_flag	1 = delta_pic_order_cnt[0] and delta_pic_order_cnt[1] are not present in the slice headers of the sequence and are assumed to be 0. 0 = delta_pic_order_cnt[0] is present in the slice headers of the sequence, and that delta_pic_order_cnt[1] can be present in the slice headers of the sequence.
gaps_in_frame_num_value_allowed_flag	Specifies the allowed values of frame_num and the decoding process in case of an inferred gap between values of frame_num.
frame_mbs_only_flag	1 = Every coded picture of the coded video sequence is a coded frame containing only frame macroblocks. 0 = Coded pictures of the coded video sequence can be coded fields or coded frames.
mb_adaptive_frame_field_flag	1 = Specifies the possible use of switching between frame and field macroblocks within frames. 0 = Specifies no switching between frame and field macroblocks within a picture. When the mb_adaptive_frame_field_flag is not present, it is assumed to be 0.
direct_8x8_inference_flag	Specifies the method used in the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8. When frame_mbs_only_flag is equal to 0, direct_8x8_inference_flag is 1.
sps_reserved	Reserved. Must be 0.

Unless indicated otherwise, the parameters in Table 4 correspond to the same-named fields in the H.264/ACV1 specification.

Table 4 pps_info Structure Flags

Parameter	Description
entropy_coding_mode_flag	Selects the entropy decoding method. 0 = Exp-Golomb coded or CAVLC. 1 = CABAC.
pic_order_present_flag	1 = Picture-order count-related syntax elements are in the slice headers. 0 = Picture-order count-related syntax elements are not in the slice headers.
weighted_pred_flag	1 = Weighted prediction applied to P and SP slices. 0 = Weighted prediction not applied to P and SP slices.
weighted_bipred_idc	The following are valid values. 0 = the default weighted prediction is applied to B slices. 1 = explicit weighted prediction is applied to B slices. 2 = implicit weighted prediction is applied to B slices.
deblocking_filter_control_present_flag	1 = The syntax elements controlling the characteristics of the deblocking filter is in the slice header. 0 = The syntax elements controlling the characteristics of the deblocking filter is not in the slice headers, and their inferred values are in effect.
constrained_intra_pred_flag	1 = Constrained intra prediction: prediction of macroblocks coded using Intra-macroblock prediction modes only uses residual data and decoded samples from I or SI macroblock types. 0 = Intra prediction allows usage of residual data and decoded samples of neighboring macroblocks coded using Inter-macroblock prediction modes.
redundant_pic_cnt_present_flag	1 = The redundant_pic_cnt syntax element is in all slice headers, data partitions B, and data partitions C that refer (either directly or by association with a corresponding data partition A) to the picture parameter set. 0 = The redundant_pic_cnt syntax element is not in slice headers, data partitions B, or data partitions C that refer (either directly or by association with a corresponding data partition A) to the picture parameter set.
transform_8x8_mode_flag	1 = The 8x8 transform decoding process can be in use. 0 = The 8x8 transform decoding process is not in use. When the transform_8x8_mode_flag is not present, it is assumed to be 0.
pps_reserved	Reserved field. Must be 0.
picture_structure	Specifies the picture type. 0 = Frame. 1 = Top field. 2 = Bottom field.
chroma_format	Specifies the chroma sampling relative to luma sampling. 0 = monochrome 1 = 4:2:0 2 = 4:2:2 3 = 4:4:4 When chroma_format is not present, it is assumed to be equal to 1 (4:2:0 chroma format).

Table 4 pps_info Structure Flags (Cont.)

Parameter	Description
bit_depth_luma_minus8	Specifies the bit depth of the samples of the luma array and the value of the luma quantization parameter range offset. BitDepthY = 8 + bit_depth_luma_minus8 QpBdOffsetY = 6 * bit_depth_luma_minus8 When not present, it is assumed to be equal to 0. The value must be from 0 to 4, inclusive.
bit_depth_chroma_minus8	Specifies the bit depth of the samples of the chroma arrays and the value of the chroma quantization parameter range offset, as specified by: BitDepthC = 8 + bit_depth_chroma_minus8 QpBdOffsetC = 6 * (bit_depth_chroma_minus8 + residual_colour_transform_flag) When bit_depth_chroma_minus8 is not present, it is assumed to be equal to 0. The value must be in the range 0 to 4, inclusive.
log2_max_frame_num_minus4	Specifies the value of the variable MaxFrameNum that is used in *frame_num* related derivations as follows: MaxFrameNum = 2 power of (log2_max_frame_num_minus4 + 4) The value must be in the range 0 to 12, inclusive.
pic_order_cnt_type	Specifies the method to decode picture order count (POC). The value must be in the range 0 to 2, inclusive.
log2_max_pic_order_cnt_lsb_minus 4	Specifies the value of the variable MaxPicOrderCntLsb that is used in the decoding process for picture order count. See the H.264 specification, subclause 8.2.1: MaxPicOrderCntLsb = $2^{(log2\_max\_pic\_order\_cnt\_lsb\_minus4 + 4)}$ The value must be in the range 0 to 12, inclusive.
num_ref_frames	Specifies the maximum number of short-term and long-term reference frames, complementary reference field pairs, and non-paired reference fields that can be used by the decoding process for inter prediction of any picture in the sequence. This function also determines the size of the sliding window operation. The value must be in the range 0 to MaxDpbSize, inclusive.
reserved_8bit	Reserved. Must be 0. No correspondence with H.264 specification.
pic_init_qp_minus26	Specifies the initial value minus 26 of SliceQPY for each slice. The initial value is modified at the slice layer when a non-zero value of slice_qp_delta is decoded; it is modified further when a non-zero value of mb_qp_delta is decoded at the macroblock layer. The value must be in the range (26 + QpBdOffsetY) to +25, inclusive.
pic_init_qs_minus26	Specifies the initial value minus 26 of SliceQSY for all macroblocks in SP or SI slices. The initial value is modified at the slice layer when a non-zero value of slice_qs_delta is decoded. The value must be in the range of -26 to +25, inclusive.
chroma_qp_index_offset	Specifies the offset to be added to QPY and QSY for addressing the table of QPC values for the Cb chroma component. The value must be in the range of -12 to +12, inclusive.
second_chroma_qp_index_offset	Specifies the offset to be added to QPY and QSY for addressing the table of QPC values for the Cr chroma component. The value must be in the range of 12 to +12, inclusive. When second_chroma_qp_index_offset is not present, it is assumed to be equal to avc_chroma_qp_index_offset.

Table 4 pps_info Structure Flags (Cont.)

Parameter	Description
num_slice_groups_minus1	Specifies the number of slice groups for a picture minus 1. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. 0 = for H.264 main and high profiles. 0-7 = for H.264 baseline profile.
slice_group_map_type	Specifies how the mapping of slice group map units to slice groups is coded. The value must be in the range of 0 to 6, inclusive.
num_ref_idx_l0_active_minus1	Specifies the maximum reference index for reference picture list 0 to be used to decode each slice of the picture in which list 0 prediction is used when num_ref_idx_active_override_flag is 0 for the slice. When MbaffFrameFlag is 1, num_ref_idx_l0_active_minus1 is the maximum index value for the decoding of frame macroblocks, and 2 * num_ref_idx_l0_active_minus1 + 1 is the maximum index value for the decoding of field macroblocks. The value must be in the range of 0 to 31, inclusive.
num_ref_idx_l1_active_minus1	It has the same semantics as avc_num_ref_idx_l0_active_minus1 with l0 and list 0 replaced by l1 and list 1, respectively. The semantics are the same as for num_ref_idx_l0_active_minus1.
slice_group_change_rate_minus1	Specifies the variable SliceGroupChangeRate, which determines the multiple (in number of slice group map units) by which the size of a slice group can change from one picture to the next. The value must be in the range of 0 to PicSizeInMapUnits – 1, inclusive. The SliceGroupChangeRate variable is specified as follows: SliceGroupChangeRate = slice_group_change_rate_minus1 + 1
reserved_16bit	Reserved. Must be 0.
scaling_lists_4x4 [6][16]	4x4 quantization matrix data in zig-zag scan order.
scaling_lists_8x8 [2][64]	The 8x8 quantization matrix data in zig-zag scan order.
frame_num	The field is used as an identifier for pictures. In H26.4, it is represented by log2_max_frame_num_minus4 + 4 bits in the bitstream.
frame_num_list[16]	Not used. Must be 0.
curr_field_order_cnt_list[2]	curr_field_order_cnt_list[0] corresponds to *TopFieldOrderCnt* in the H.264 specification. curr_field_order_cnt_list[1] corresponds to *BottomFieldOrderCnt* in the H.264 specification. Determines the initial picture ordering for reference pictures in the decoding of B slices to represent picture order differences between frames or fields - for motion vector derivation in temporal direct mode, - for implicit mode weighted prediction in B slices, and - for decoder conformance checking.
field_order_cnt_list[16][2]	Reserved. must be 0.
intra_flag	Specifies the prediction mode type in a frame/field. 1 = Picture is coded in Intra prediction mode. It supposes that I-frames are coded in the Intra prediction mode only. 0 = the flag specifies that the picture can be coded in Inter prediction mode.
reference	Specifies whether this picture is used as the reference picture. 1 = Reference picture. 0 = Non-reference picture.

Table 5 mvc Structure

Parameter	Description
numViews	Number of coded views.
viewID0	Base view ID.
mvcElements []	mvcElement_t structure array, allocation Must be numViews -1.

Table 6 mvcElement_t Structure

Parameter	Description
viewOrderIndex	View order index.
viewID	ViewID of each view.
numOfAnchorRefsInL0	Number of Anchor inter-views in L0.
viewIDofAnchorRefsInL0[15]	Anchor inter-view viewID in L0.
numOfAnchorRefsInL1	Number of Anchor inter-views in L1.
viewIDofAnchorRefsInL1[15]	Anchor inter-view viewID in L1.
numOfNonAnchorRefsInL0	Number of Non-anchor inter-views in L0.
viewIDofNonAnchorRefsInL0[15]	Non-anchor inter-view viewID in L0.
numOfNonAnchorRefsInL1	Number of Non-anchor inter-views in L1.
viewIDofNonAnchorRefsInL1[15]	Non-anchor inter-view viewID in L1.
reserved [128]	Not used, always 0.

4.2.2 VC-1 **picture_parameter_2** Structure

```
typedef struct
{
   union{
      struct {
         unsigned int      postprocflag         : 1;
         unsigned int      pulldown             : 1;
         unsigned int      interlace            : 1;
         unsigned int      tfcntrflag           : 1;
         unsigned int      finterpflag          : 1;
         unsigned int      sps_reserved1        : 1;
         nsigned int       psf                  : 1;
         unsigned int      second_field         : 1;
         unsigned int      sps_reserved2        : 24;
      } sps_flag;
      unsigned int     flag;
   } sps_info;

   union {
      struct {
         unsigned int      panscan_flag         : 1;
         unsigned int      refdist_flag         : 1;
         unsigned int      loopfilter           : 1;
         unsigned int      fastuvmc             : 1;
         unsigned int      extended_mv          : 1;
         unsigned int      dquant               : 2;
         unsigned int      vstransform          : 1;
         unsigned int      overlap              : 1;
```

```
            unsigned int      quantizer           : 2;
            unsigned int      extended_dmv        : 1;
            unsigned int      maxbframes          : 3;
            unsigned int      rangered            : 1;
            unsigned int      syncmarker          : 1;
            unsigned int      multires            : 1;
            unsigned int      reserved            : 2;
            unsigned int      range_mapy_flag     : 1;
            unsigned int      range_mapy          : 3;
            unsigned int      range_mapuv_flag    : 1;
            unsigned int      range_mapuv         : 3;
            unsigned int      vc1_pps_reserved    : 4;
         } pps_flag;
         unsigned int    flag;
      } pps_info;

      unsigned int     picture_structure;
      unsigned int     chroma_format;
      unsigned int     reserved [128];

   } VC1_picture_parameter_2;
```

The description of each field in the VC1_picture_parameter_2 data structure is given in Tables 7 to 9, below.

Table 7 sps_info Structure Flags

Parameter	Description
postprocflag	Indicates whether the syntax element POSTPROC is present in picture headers. 1 = Syntax element POSTPROC is present in picture headers. 0 = Syntax element POSTPROC is not present in picture headers.
pulldown	Indicates whether the syntax elements RPTFRM, or TFF and RFF are present in picture headers. 1 = Syntax elements RPTFRM, or TFF and RFF are present in picture headers. 0 = Not present.
interlace	1 = Individual frames must be coded using the progressive or interlace syntax. 0 = Pictures must be coded as single frames using the progressive syntax.
tfcntrflag	It is a frame counter flag. 1 = Indicates that the syntax element TFCNTR must be present in the advanced profile picture headers. 0 = Indicates that TFCNTR must not be present in the picture header.
finterpflag	A frame interpolation flag that specifies if the syntax element INTERPFRM is present in the picture header. 1 = INTERPFRM is present in picture headers. 0 = INTERPFRM is not present in picture headers.
sps_reserved1	Corresponds to the *RESERVED* field in the VC-1 specification. This field must be set to 1. This one-bit flag corresponds to the one-bit syntax element Reserved Advanced Profile Flag defined in the VC-1 specification. The value 0 is SMPTE reserved. Must be set to 1. Reserved Advanced Profile Flag.
psf	Specifies the video source. 1 = The video source was Progressive Segmented Frame (PsF), and the display process treats the decoded frames (field-pairs) as progressive. 0 = The display process can treat the decoded frames (field-pairs) according to the value of the *INTERLACE* syntax element.
second_field	Specifies whether the picture is the second field. 0 = The picture is a frame or the first field. 1 = The picture is the second field.
sps_reserved2	Reserved. Must be 0.

In Table 8, unless indicated otherwise, the parameters correspond to the same-named field in the VC-1 specification.

Table 8 pps_info Structure Flags

Parameter	Description
panscan_flag	1 = specifies that pan scan regions are present for pictures within that entry point segment. The pan scan region is a sub-region of the display region which may be used as an alternative presentation format. The most common application is to display a 4:3 sub-region of 16:9 content. 0 = specifies that pan scan regions are not present.
refdist_flag	A Reference Frame Distance Flag. 1 = specifies that the REFDIST syntax element is present in II, IP, PI or PP field picture headers. 0 = the REFDIST syntax element is not present.
loopfilter	1 = specifies that loop filtering is enabled. 0 = specifies that loop filtering is not enabled. If the stream PROFILE is Simple, the LOOPFILTER must be 0.
fastuvmc	A Fast UV Motion Compensation Flag. It controls the subpixel interpolation and rounding of color-difference motion vectors. 1 = specifies that the color-difference motion vectors that are at quarter pel offsets are rounded to the nearest half or full pel positions. 0 = no special rounding or filtering is done for color-difference. If the stream Profile is Simple, this must be 0.
extended_mv	The Extended Motion Vector Flag. Specifies whether extended motion vectors are enabled (1) or disabled (0). This bit must be set to 0 for the Simple Profile. For the Main Profile, the extended motion vector mode must indicate the possibility of extended motion vectors in P and B pictures.
dquant	Specifies whether or not the quantization step size can vary within a frame. 0 = only one quantization step size (i.e. the frame quantization step size) is used per frame. 1 or 2 = the quantization step size may vary within the frame. In Simple profile, DQUANT must be 0. In the Main profile, if MULTIRES = 1, DQUANT must be 0.
vstransform	Specifies whether variable-sized transform coding is enabled for the sequence. 1 = variable-sized transform coding must be enabled. 0 = variable-sized transform coding must not be enabled.
overlap	Specifies whether Overlapped Transforms are used. 1 = Overlapped Transforms can be used. 0 = Overlapped Transforms are not used.
quantizer	Specifies the quantizer used for the sequence. 0 = Quantizer implicitly specified at frame level. 1 = Quantizer explicitly specified at frame level. 2 = Nonuniform quantizer used for all frames. 3 = Uniform quantizer used for all frames.
extended_dmv	1 = specifies that extended differential motion vector range is signaled at the picture layer for the P and B pictures within the entry point segment. 0 = specifies that extended differential motion vector range is not signaled.

Table 8 pps_info Structure Flags (Cont.)

Parameter	Description
maxbframes	Specifies the maximum number of consecutive B frames between I or P frames. 0 = there are no B frames in the sequence. 0-7 = this number of B Frames can be present in the sequence.
rangered	Specifies whether range reduction is used for each frame. 1 = each frame header must contain a syntax element, RANGEREDFRM, that indicates whether range reduction is used for that frame. 0 = the syntax element RANGEREDFRM is not present, and range reduction must not used. RANGERED must be set to zero in Simple profile.
syncmarker	Indicates whether synchronization markers can be present in the bitstream. In the main profile, the synchronizations markers can be present if SYNCMARKER = 1; they can not be present if SYNCMARKER = 0. In simple profile, this must be 0.
multires	A Multiresolution Coding flag that specifies whether the frames can be coded at smaller resolutions than the specified frame resolution. Resolution changes are allowed only on I pictures. 1 = the frame level RESPIC syntax element must be present, indicating the resolution for that frame. 0 = RESPIC must not be present.
reserved1	The field corresponds *Reserved6* field in the VC-1 specification. Controls the video stream. Must be set to 1.
range_mapy_flag	The Range Mapping Luma Flag. Specifies whether RANGE_MAPY is present in the entry header. 1 = RANGE_MAPY is present in the entry header. 0 = RANGE_MAPY is not present in the entry header.
range_mapy	The Range Mapping Luma value must be present if range_mapy_flag is set to 1. The value of range_mapy must be in the range of 0 to 7, inclusive. If this syntax element is present, the luma components of the decoded pictures within the entry point segment must be scaled according to the formula: Y[n] = CLIP ((((Y[n] – 128) * (RANGE_MAPY + 9) + 4) >> 3) + 128);
range_mapuv_flag	The Range Mapping Color-Difference Flag. Specifies whether RANGE_MAPUV is present in the entry header. 1 = RANGE_MAPUV is present in within the entry header. 0 = RANGE_MAPUV is not present in within the entry header.
range_mapuv	The Range Mapping Color-Difference value must be present if range_mapy_flag is set to 1. The value of range_mapuv must be in the range of 0 to 7, inclusive. If this syntax element is present, the color-difference components of the decoded pictures within the entry point segment must be scaled according to the formula: Cb[n] = CLIP ((((Cb[n] – 128) * (RANGE_MAPUV + 9) + 4) >> 3) + 128); Cr[n] = CLIP ((((Cr[n] – 128) * (RANGE_MAPUV + 9) + 4) >> 3) + 128);
pps_reserved2	Reserved. Must be 0.

In Table 9, lists and briefly describes the common fields in the VC1_picture_parameter_2 data structure.

Table 9 Common Fields in VC1_picture_parameter_2 Structure

Parameter	Description
picture_structure	Specifies the type of picture: 1 = top field. 2 = bottom field. 3 = frame.
chroma_format	Specifies the chroma sampling relative to the luma sampling. 0 = monochrome. 1 = 4:2:0. 2 = 4:2:2. 3 = 4:4:4. When chroma_format is not present, it is assumed to be 1 (4:2:0 chroma format).
reserved [128]	Reserved. Must be 0.

4.3 OVD_BITSTREAM_DATA and OVD_SLICE_DATA_CONTROL

This section describes the slice data and the slice control data layout that the application must construct for the OVD interface.

4.3.1 OVD_SLICE_DATA_CONTROL Structure

```
typedef struct
{
  unsigned int            SliceBitsInBuffer;
  unsigned int            SliceDataLocation;
  unsigned int            SliceBytesInBuffer;
  unsigned int            reserved[5];

} ovd_slice_data_ctrl;
```

4.3.2 Bitstream Data Buffer and Slice Data Control Data Layout

OVD_BITSTREAM_DATA contains blocks of compressed bitstream data (see Figure 3). The Decoder (host) stores the data blocks size/location information in OVD_SLICE_DATA_CONTROL. Every data block has its own control data structure. OVD_BITSTREAM_DATA must be 128 byte aligned.

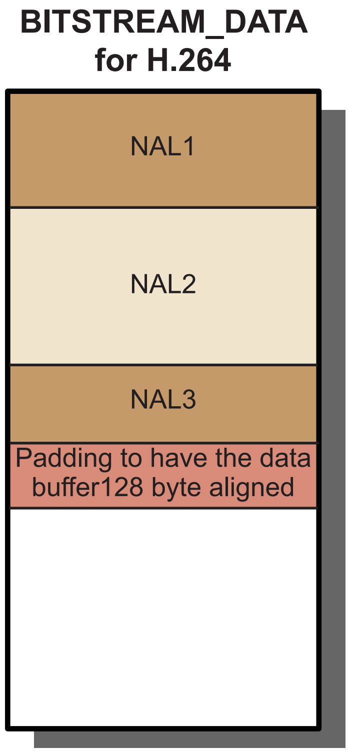

Figure 3 OVD_BITSTREAM_DATA and OVD_SLICE_DATA_CONTROL Example for H.264

Contact

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA, 94088-3453
Phone: +1.408.749.4000

For Stream Computing:
URL: www.amd.com/stream
Questions: streamcomputing@amd.com
Developing: developer.amd.com/support
Forum: www.amd.com/streamdevforum