ISSUE TO	COPY NO.

XVBA
AMD's Linux Video Acceleration

Rev 0.74.01-AES-2

AMD Embedded Solutions

1. Introduction

This document contains AMD's video pipeline API for Linux (GPU decode acceleration). New API is named XVBA: Xv Bitstream Acceleration. This document explains the XVBA infrastructure role in Linux with and without decode GPU acceleration.

XVBA API idea has similarity to XvMC API (version 1.0). XVBA **does not** support the XvMC MPEG2 decode.

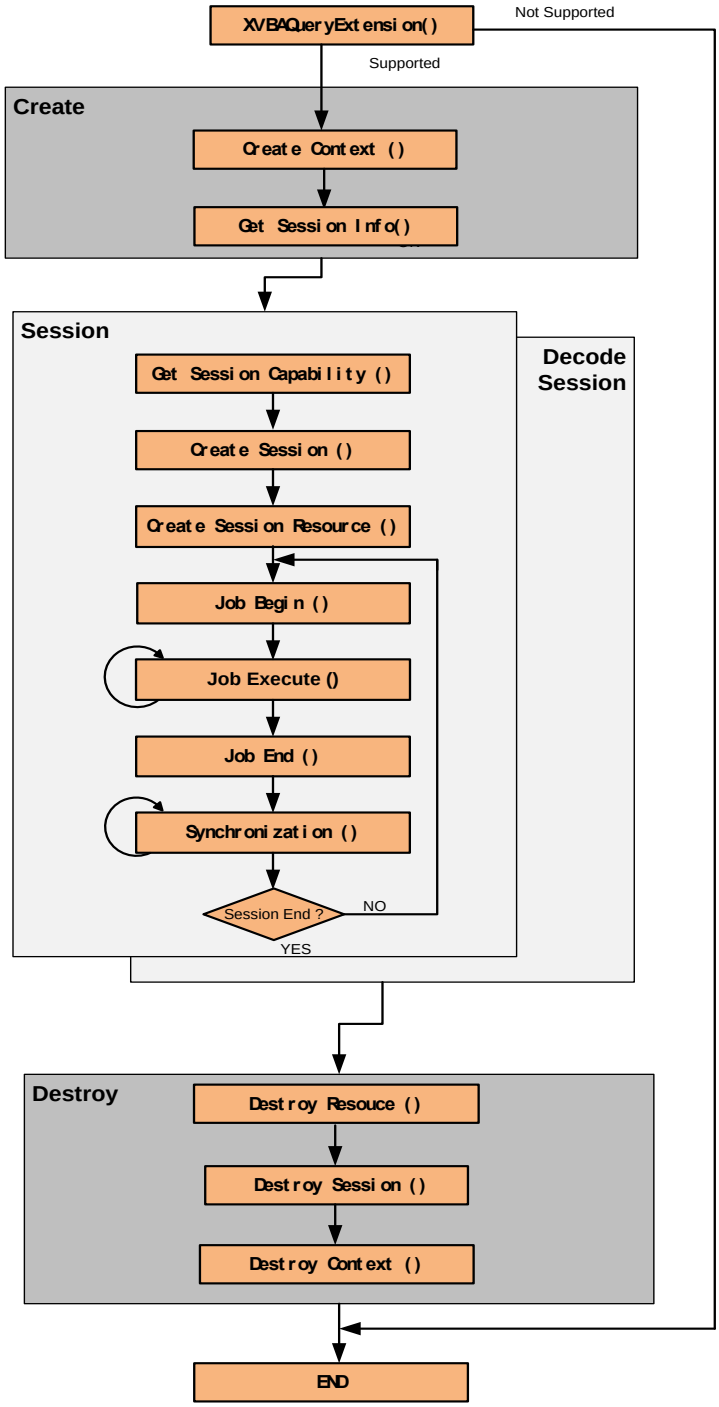
XVBA API design goals and highlights:

- XVBA MPEG2 with new compressed data buffers layout (bit layout supported by AMD HW)
- XVBA should be simple to implement for decoder that already has XvMC API working.
- XVBA supports bitstream decode GPU based acceleration– first revision supports h.264 and VC-1.MPEG2 is supported at IDCT level.
- XVBA is extendable for new codecs. Only bitstream level decode acceleration is planned to be supported in future revisions.

2. XVBA High Level System Overview

2.1 XVBA High Level Diagram

The following diagram shows the high level view of XVBA pipeline.



Definition

- **Context**: Application created driver context.
- **Session**: XVBA capability entity.

- **Job**: GPU job.
- **Session capability**: GPUfeature exposed through driver to application.

2.2 XVBA Design Notes

What is new in XVBA compared to XvMC:

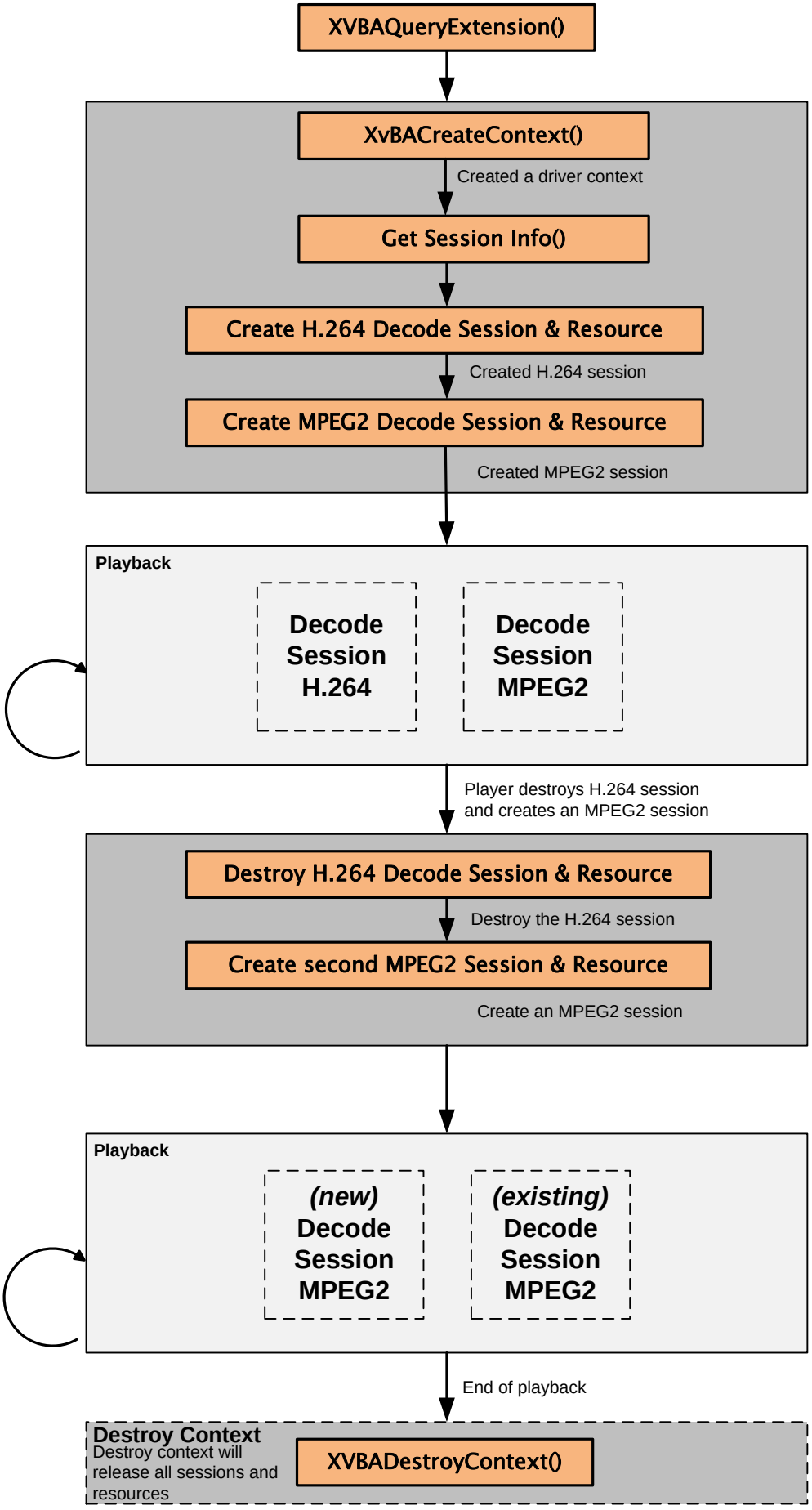
- New API functions for bitstream decode
- New data structures and new bitstream compressed data structures
- New functions: Start and End picture decoding. These function are important for driver to set decode jobs correctly on the accelerator.
- Multi decode session within one context
- XVBA adopts concept of capability and session
- Session can be added and removed without destroying (and re-creating) context
- XVBA is expandable for new capabilities

Notes on XVBA API

- Only 1 drawable is supported in driver context
- Context contains 1 or more sessions
- Sessions can be removed and added to context without destroying the context
- Session creates and owns surfaces
- Context can destroy all sessions and release its resources

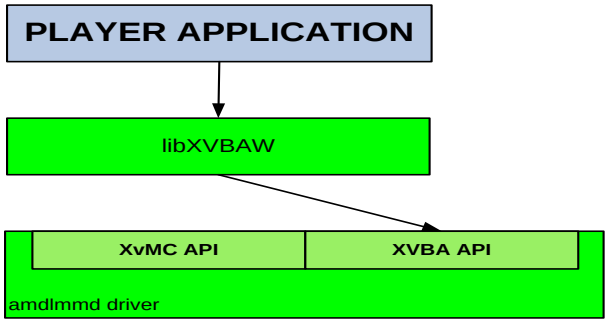
2.3 XVBA Multi-Sessions Usage Scenario

For XVBA pipeline, application is capable of creating multiple sessions within one XVBA context. Multi-sessions can be either the combination of different sessions or same multiple sessions. The following diagram shows the high level view of multi-sessions usage scenario.


XVBAQueryExtension()
XvBACreateContext()
Created a driver context
Get Session Info()
Create H.264 Decode Session & Resource
Created H.264 session
Create MPEG2 Decode Session & Resource
Created MPEG2 session
Playback
Decode Session H.264
Decode Session MPEG2
Player destroys H.264 session and creates an MPEG2 session
Destroy H.264 Decode Session & Resource
Destroy the H.264 session
Create second MPEG2 Session & Resource
Create an MPEG2 session
Playback
(new) Decode Session MPEG2
(existing) Decode Session MPEG2
End of playback
Destroy Context
Destroy context will release all sessions and resources
XVBADestroyContext()

2.4 Application Components Dependency Diagram

Figure below shows the structure diagram between player application and the driver using XVBA API.



- Application is linked to libXVBAW library which provides XVBA interface.
- *libXVBAW* library is responsible to load the corresponding *amdlmmd* driver and to retrieves all necessary pointers from them.

3. XVBA API Description and Data Structures

3.1 XVBA Video Playback Pipeline

XVBA allows player to use GPU in the following scenarios:

- Hardware Decode with OpenGL (full hardware video playback pipeline)
- Hardware Decode with X11 or XVideo presentation (not shown)

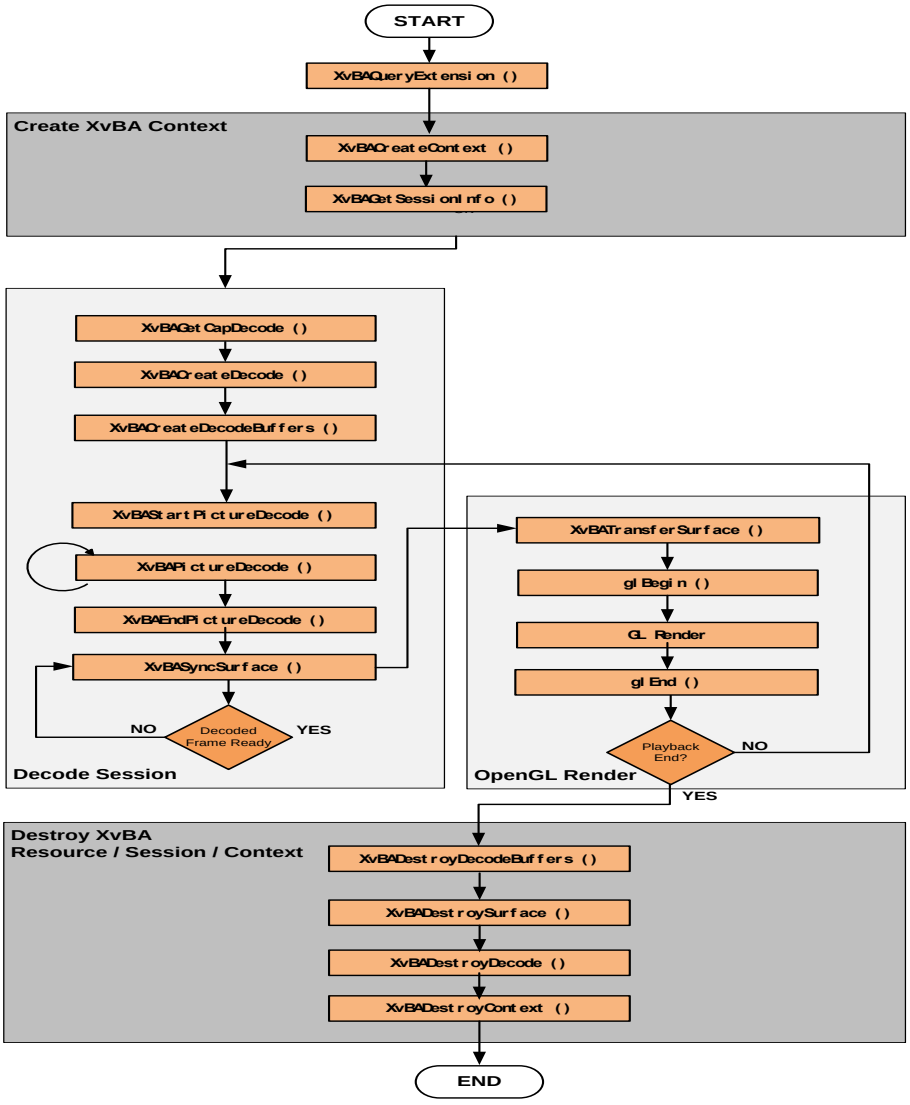


Diagram of Video Playback Pipeline Using XVBA API

3.2 XVBAQueryExtension

Application has to check first if XVBA is available. If XVBAQueryExtension() returns TRUE host queries for the XVBA caps; else if this function returns FALSE application may try XvMC or Xv.

```
Bool
XVBAQueryExtension
(
        Display *display,               /*in*/
        int     *version                /*out*/
);
```

display – connection to the server.

version – returned XVBA version = XVBA_VERSION_MAJOR << 16) | XVBA_VERSION_MINOR

Returns: True if XVBA is supported with XVBAVersion, False otherwise.

3.3 Create & Destroy XVBA Context

- To use XVBA, application must create a XVBA context first
- Context is based on Display and Drawable.
- Context can have only 1 Drawable.
- Context can have 1 or more sessions.

3.3.1 XVBACreateContext

```
typedef struct {
   unsigned int    size;               //structure size

   Display         *display;
   Drawable        draw;

} XVBA_Create_Context_Input;
```

```
typedef struct {
   unsigned int    size;               //structure size

   void           *context;

} XVBA_Create_Context_Output;
```

```
Status
XVBACreateContext (
   XVBA_Create_Context_Input   *create_context_input,    /*in*/
   XVBA_Create_Context_Output  *create_context_output    /*out*/
);
```

Driver creates a context and returns a pointer to its context.

Errors:

BadValue – invalid input values: display or drawable.

3.3.2 XVBADestroyContext

- Destroying context will release all sessions in this context and resources own by each session

```
Status
XVBADestroyContext (
        void    *context
);
```

*context - Pointer to the XVBA driver context structure.

Errors:

BadContext - XVBAContext is not valid.

3.4 Query Session Info

Application has to query for session info before session creation in XVBA. In XVBA version 1.0, the following sessions are exposed:

- Decode (H.264, VC-1, MPEG2)

3.4.1 XVBAGetSessionInfo

```
typedef struct {
   unsigned int    size;              //structure size
   void            *context;

} XVBA_GetSessionInfo_Input;
```

```
typedef struct {
   unsigned int    size;                    //structure size

   unsigned int    getcapdecode_output_size; // 0 = Decode not supported, non zero value = Decode session is
                                                supported and this value is used for XVBAGetCapDecode output
                                                struct size
   unsigned int    xvba_gsio_reserved_0;     // Not used by XVBA
   unsigned int    xvba_gsio_reserved_1;     // Not used by XVBA

} XVBA_GetSessionInfo_Output;
```

```
Bool
XVBAGetSessionInfo
(
        XVBA_GetSessionInfo_Input      *get_session_info_input,
        XVBA_GetSessionInfo_Output     *get_session_info_output
);
```

Returns: True if capability list was successfully created, False otherwise.

3.5 Create & Destroy Session Resource

3.5.1 Create and Destroy Surface

3.5.1.1 XVBACreateSurface

- This function creates a surface within the specified context.
- Surface can be used by other sessions within the same context.
- Surface is owned by session that created it.
- Surface can be destroyed only by session that created it.
- XVBACreateSurface allocates 1 surface at the time.

```
typedef struct
{
  unsigned int          size;

  void                  *session;
  unsigned int          width;
  unsigned int          height;
  XVBA_SURFACE_FORMAT   surface_type;

} XVBA_Create_Surface_Input;
```

```
typedef struct
{
  unsigned int     size;
  void             *surface;      // Pointer to XVBASurface

} XVBA_Create_Surface_Output;
```

```
Status
XVBACreateSurface(
        XVBA_Create_Surface_Input      *create_surface_input,
        XVBA_Create_Surface_Output     *create_surface_output
);
```

Errors:
BadValue – invalid data
BadAlloc – there are insufficient resources to complete this operation.

3.5.1.2 XVBACreateGLSharedSurface

- This function creates a XVBA shared surface holder for OpenGL texture buffer within XVBA pipeline.

```
typedef struct
{
  unsigned int          size;

  void                  *session;
  void                  *glcontext;
  unsigned int          gltexture;

} XVBA_Create_GLShared_Surface_Input;
```

```
typedef struct
{
  unsigned int     size;
  void             *surface;      // Pointer to XVBASurface

} XVBA_Create_GLShared_Surface_Output;
```

```
Status
XVBACreateGLSharedSurface(
        XVBA_Create_GLShared_Surface_Input     *create_glshared_surface_input,
        XVBA_Create_GLShared_Surface_Output    *create_glshared_surface_output
);
```

Errors:

BadValue – invalid data

BadAlloc – there are insufficient resources to complete this operation.

3.5.1.3 XVBADestroySurface

```
Status
XVBADestroySurface(
        void    *surface
);
```

surface – surface to be destroyed.

Errors:

BadSurface – XVBASurface is not valid.

3.5.2 Create and Destroy Compressed Data Buffers (decode)

```
typedef enum
{
    XVBA_NONE = 0,
    XVBA_PICTURE_DESCRIPTION_BUFFER,
    XVBA_DATA_BUFFER,
    XVBA_DATA_CTRL_BUFFER,
    XVBA_QM_BUFFER

} XVBA_BUFFER;
```

```
typedef struct
 {
    unsigned int    size;                   //structure size
    XVBA_BUFFER     buffer_type;
    int             buffer_size;            //allocated size of data in bytes
    void            *bufferXVBA;            //pointer to XVBA decode data buffer
    int             data_size_in_buffer;    //Used in Decode call only
    int             data_offset;            //Used in Decode call only
    void            *appPrivate;            //used only by application to store pointer to its private data.

} XVBABufferDescriptor;
```

3.5.2.1 XVBACreateDecodeBuffers

```
typedef struct
{
        unsigned int                size;    //structure size

        void                        *session;
        XVBA_BUFFER                 buffer_type;
        unsigned int                num_of_buffers;

} XVBA_Create_DecodeBuff_Input;
```

```
typedef struct
{
        unsigned int                size;    //structure size

        unsigned int                num_of buffers_in_list;
        XVBABufferDescriptor        *buffer_list;

} XVBA_Create_DecodeBuff_Output;
```

```
Status
XVBACreateDecodeBuffers (
    XVBA_Create_DecodeBuff_Input      *create_decodebuff_input,
    XVBA_Create_DecodeBuff_Output     *create_decodebuff_output
);
```

Errors:

BadAlloc – There are insufficient resources to complete the operation.
BadValue – bad input data

3.5.2.2 XVBADestroyDecodeBuffers

```
typedef struct
{
        unsigned int                size;

        void                        *session
        unsigned int                num_of_buffers_in_list;
        XVBABufferDescriptor        *buffer_list;

} XVBA_Destroy_Decode_Buffers_Input;
```

session – pointer to XVBASession.
num_of_buffer_in_list – number of decode compressed data buffers to be released
bufferList – array of XVBA_BUFFER_DESCRIPTOR structures

Frees resources allocated for decode (compressed data buffers).

```
Status
XVBADestroyDecodeBuffers (
        XVBA_Destroy_Decode_Buffers_Input     *buffer_list
);
```

Errors:
BadValue – bad input data

Notes:

- If XVBADestroyDecode() is called to destroy a decode session and it will automatically release all resources owned session. There is no need to call an XVBADestroyDecodeBuffers or XVBADestroySurface() separately before releasing the entire session.
- Similar for XVBADestroyContext() all allocated resources will be released; no need to call XVBA destroy functions separately.
- Application can use this function to release decode buffers without destroying the session.

3.6 Decode Session APIs

Every picture decode starts with one XVBAStartDecodePicture() fn call and ends with a single XVBAEndDecodePicture(). In between these two function calls, XVBA host can call multiple times XVBADecodePicture() to submit decode data buffers to driver.

3.6.1 Query Decode Capability

3.6.1.1 XVBAGetCapDecode

XVBACap structure defines capability:

```
// XVBADecodeCap capability_id
typedef enum
{
         XVBA_H264  = 0x100,          /*bitstream level of acceleration*/
         XVBA_VC1,                    /*bitstream level of acceleration*/
         XVBA_MPEG2_IDCT,             /*iDCT and motion compensation level of acceleration*/
         XVBA_MPEG2_VLD               /*bitstream level of acceleration*/
} XVBA_CAPABILITY_ID;

// XVBADecodeCap flag
typedef enum
{
         XVBA_NOFLAG        = 0;

         XVBA_H264_BASELINE,
         XVBA_H264_MAIN,
         XVBA_H264_HIGH,

         XVBA_VC1_SIMPLE,
         XVBA_VC1_MAIN,
         XVBA_VC1_ADVANCED,

} XVBA_DECODE_FLAGS;

typedef struct {
   unsigned int              size;               //structure size

   XVBA_CAPABILITY_ID        capability_id;
   XVBA_DECODE_FLAGS         flags;
   XVBA_SURFACE_FORMAT       surface_tpye;

} XVBADecodeCap;
```

capability_id – description of acceleration level.
flags – defines for additional information about capability
surface_type – fourcc YUV or RGB supported with this capability.

```
typedef enum
{
         XVBA_FRAME = 0;
         XVBA_TOP_FIELD,
         XVBA_BOTTOM_FIELD,
} XVBA_SURFACE_FLAG;

typedef struct {
   unsigned int              size;                //structure size

   XVBA_SURFACE_FORMAT       surface_tpye;
   XVBA_SURFACE_FLAG         flag;

} XVBA_GetSurface_Target;
```

```
typedef struct {
   unsigned int    size;              //structure size
   void            *context;

} XVBA_GetCapDecode_Input;
```

```
typedef struct {
   unsigned int    size;              //structure size

   unsigned int    num_of_decodecaps;
   XVBADecodeCap   decode_caps_list [];

   unsigned int              num_of_getsurface_target;
   XVBA_GetSurface_Target    getsurface_target_list [];

} XVBA_GetCapDecode_Output; // this structure size should match on the value returned from GetSessionInfo
()
```

```
Bool
XVBAGetCapDecode (
         XVBA_GetCapDecode_Input         *decodecap_list_input,
         XVBA_GetCapDecode_Ouptut        *decodecap_list_output
);
```

Returns: True if capability list was successfully created, False otherwise.

Example of Reporting Decode Capability

Example: if accelerator supports various decode targets in capability it will report it separately in preference order:

```
//Capability 1
{
   capability_id = XVBA_H264;
   flags         = XVBA_H264_BASELINE;
   surface_type  = NV12;
}
//Capability 2
{
   capability_id = XVBA_H264;
   flags         = XVBA_H264_MAIN;
   surface_type  = NV12;
}
//Capability 3
{
   capability_id = XVBA_H264;
   flags         = XVBA_H264_HIGH;
   surface_type  = NV12;
}
//Capability 2
```

```
{
   capability_id = XVBA_H264;
   flags         = XVBA_H264_BASELINE;
   surface_type  = YV12;
}
//Capability 2
{
   capability_id = XVBA_h264;
   flags         = XVBA_H264_MAIN;
   surface_type  = YV12;
}
…
```

In this example driver signals to the application that NV12 surface is the most preferred for 4:2:0 decode.

Note: in current AMD accelerators only NV12 decode target is supported. Above example does not imply other YUV decoder target availability and/or AMD accelerator product roadmap.

3.6.2 Create/Destroy XVBA Decode Session

- Context can have 1 or more decode sessions.
- Session must be created before resource creation (surface or compressed buffer)
- Session owns the surfaces it created.
- Surfaces can be shared with other session within the context

3.6.2.1 XVBACreateDecode

```
typedef struct {
   unsigned int              size;              //structure size
   unsigned int              width;             // Decoded video width
   unsigned int              height;            // Decoded video height
   void                      *context;
   XVBADecodeCap             *decode_cap;

} XVBA_Create_Decode_Session_Input;


typedef struct {
   unsigned int    size;              //structure size
   void            *session;          // Pointer to the created decode session

} XVBA_Create_Decode_Session_Output;
```

```
Status
XVBACreateDecode (
   XVBA_Create_Decode_Session_Input    *create_decode_session_input,
   XVBA_Create_Decode_Session_Output   *create_decode_session_output
);
```

Errors:

BadValue – invalid value input size or capability.

BadContext – The XVBAContext is not valid.

3.6.2.2 XVBADestroyDecode

- Destroys specified session.

- Destroying a session will release all allocated resources (surfaces, compressed decode data buffers for this session, etc.)

```
Status
XVBADestroyDecode (
        void    *session
);
```

Errors:

BadValue – invalid session

3.6.3 Decode Acceleration Functions

3.6.3.1 XVBAStartDecodePicture

```
typedef struct
{
        unsigned int       size;              //structure size

        void               *session;
        void               *target_surface;

} XVBA_Decode_Picture_Start_Input;
```

session – pointer to XVBASession.

target_surface – decode target

```
Status
XVBAStartDecodePicture (
    XVBA_Decode_Picture_Start_Input            *decode_picture_start
);
```

Errors:

BadSurface – target surface is not valid.

BadValue – bad XVBASession

3.6.3.2 XVBADecodePicture

XVBADecodePicutre is the function used by application to submit decode compressed data buffers to driver.

```
typedef struct
{
        unsigned int                size;     //structure size

        void                        *session;
        unsigned int                num_of_buffers_in_list;
        XVBABufferDescriptor        **buffer_list;

} XVBA_Decode_Picture_Input;
```

session – pointer to the decode session.

num_of_buffers_in_list – number of decode compressed data buffers

buffer_list – array of XVBABufferDescriptor structures

```
Status
XVBADecodePicture (
    XVBA_Decode_Picture_Input          *decode_picture_input
);
```

Errors:

BadSurface – Any of the surfaces are not valid.

BadValue – bad data in decode data buffer(s) (driver and hw prescreen the compressed data buffers before processing on GPU), invalid data in XVBASession, error in num_of_buffer_in_list value

Notes: Decoder can call multiple times XVBADecodePicture() to submit XVBA compressed data buffers, however there are restrictions:

- Proper sequence call for multi XVBADecodePicture():
  - XVBAStartDecodePicture()
    - XVBADecodePicture()
    - XVBADecodePicture()
    - ….
    - XVBADecodePicture()
  - XVBAEndDecodePicture
  - XVBAStartDecodePicture()
    - XVBADecodePicture()
    - XVBADecodePicture()
    - ….
    - XVBADecodePicture()
  - XVBAEndDecodePicture()
  - …
- In single XVBADecodePicture() call application can submit only 1 buffer of each type
- Application submits only 1 `XVBA_PICTURE_DESCRIPTION_BUFFER` and `XVBA_QM_BUFFER` buffer for every picture
- Application submits `XVBA_DATA_BUFFER` and `XVBA_DATA_CTRL_BUFFER` buffers together.
- It is highly recommended that application submits all bitstream data for 1 picture in 1 `XVBA_DATA_BUFFER` for H.264/VC-1. Driver will allocate `XVBA_DATA_BUFFER` buffer big enough to accommodate the all data for single picture in one XVBADecodePicture() call.

3.6.3.3 XVBAEndDecodePicture

```
typedef struct
{
        unsigned int       size;
        void               *session;

} XVBA_Decode_Picture_End_Input;
```

When compressed data decode buffers submissions to driver for current picture are completed, host notifies driver that all data for this picture decode is sent calling XVBAEndDecodePicture().

```
Status
XVBAEndDecodePicture (
    XVBA_Decode_Picture_End_Input              *decode_picture_end_input
);
```

Errors:

BadValue – invalid data

3.6.4 Synchronization and Decode Error Query

3.6.4.1 XVBASyncSurface

- Application uses this function to check if decode job is finished before presentation
- In bitstream GPU decode (h.264 and VC-1) application can use this function to query for decode errors

```
/* XVBA decode error */
typedef enum
{
        DECODE_NO_ERROR = 0,
        DECODE_BAD_PICTURE,     //the entire picture is corrupted - all MBs are invalid
        DECODE_BAD_SLICE,       //part of the picture, slice, wasn't decoded properly - all MBs in this slice are bad
        DECODE_BAD_MB           //some MBs are not decoded properly

} XVBA_DECODE_ERROR;
```

```
typedef struct
{
   unsigned int        size;           //structure size
   XVBA_DECODE_ERROR   type;
   unsigned int        num_of_bad_mbs; //number of marcoblocks that were not properly decoded

} XVBADecdoeError;
```

```
/* Synchronization query_status_flags */
typedef enum
{
   XVBA_GET_SURFACE_STATUS   = 1,     /* get surface status; is surface still used by GPU*/
   XVBA_GET_DECODE_ERRORS             /* get decode errors for target surface*/

} XVBA_QUERY_STATUS;

typedef struct
{
   unsigned int        size;
   void                *session;
   void                *surface;
   XVBA_QUERY_STATUS   query_status_flags;

} XVBA_Surface_Sync_Input;
```

```
// define for status_flags
#define XVBA_STILL_PENDING      0x00000001      ///< surface is still used by HW
#define XVBA_COMPLETED          0x00000002      ///< HW completed job on this surface
#define XVBA_NO_ERROR_DECODE    0x00000004      ///< no decode errors
```

```
#define XVBA_ERROR_DECODE       0x00000008      ///< decode errors for queried surface

typedef struct
{
   unsigned int       size;

   unsigned int       status_flags;
   XVBADecodeError    decode_error;

} XVBA_Surface_Sync_Output;
```

```
Status
XVBASyncSurface (
        XVBA_Surface_Sync_Input        *surface_sync_input,
        XVBA_Surface_Sync_Output       *surface_sync_output
);
```

Errors:

BadData – bad input data

3.6.5 Transfer Decoded Frame Data

Application may readback the decoded frame data for Xv and OpenGL rendering. XVBA provides the XVBAGetSurface() function to copy the decoded frame data from local memory to an application supplied system memory buffer. XVBA also provides the XVBATransferSurface() function to transfer from one XVBA surface to another. For an OpenGL texture buffer, the application may call the XVBACreateGLSharedSurface() function to hold OpenGL texture buffer as XVBA surface and then call the XVBATransferSurface() function to transfer the decoded data to the OpenGL texture buffer.

3.6.5.1 XVBAGetSurface

XVBAGetSurface function supports for YV12 system memory for Xv rendering. It transfers the decoded frame data to the application supplied YV12 system memory buffer. The application can request to transfer the top field, bottom field or the whole frame to the system memory. The destination width and height must be equal to the source surface width and height. In case of transferring a field, it will be scaled to the frame size in system memory.

```
typedef struct {
    unsigned int             size;          //structure size
    void                     *session;
    void                     *src_surface;
    void                     *target_buffer;
    unsigned int             target_pitch;
    unsigned int             target_width;
    unsigned int             target_height;
    XVBA_GetSurface_Target   target_parameter;
    unsigned int             reserved [4];
} XVBA_Get_Surface_Input;
```

```
Status
XVBAGetSurface (
        XVBA_Get_Surface_Input *get_surface_input    /*in*/
);
```

Errors:

- BadValue – Invalid input data
- BadAlloc – Intermediate surface creation failed.

3.6.5.2 XVBATransferSurface

XVBATransferSurface transfers one XVBA surface to another. The application can request to transfer the top field, bottom field or whole frame of source surface to the destination surface. The destination surface width and height must be equal to the source surface width and height. In case of transferring a field, it will be scaled to the frame size in the destination.

```
typedef struct {
    unsigned int            size;          //structure size
    void                    *session;
    void                    *src_surface;      // pointer to source XVBA surface
    void                    *target_surface;   // pointer to target XVBA surface
    XVBA_SURFACE_FLAG       flag;
    unsigned int            reserved [4];
} XVBA_Transfer_Surface_Input;
```

```
Status
XVBATransferSurface (
        XVBA_Transfer_Surface_Input    *transfer_surface_input    /*in*/
);
```

Errors:

- BadValue – Invalid input data or failed.

4. Decode Data buffers

XVBA revision 1 supports 4 decode buffer types for H.264, VC-1 and MPEG2 decode:

1) XVBA_PICTURE_DESCRIPTOR_BUFFER
2) XVBA_DATA_BUFFER
3) XVBA_DATA_CTRL_BUFFER
4) XVBA_QM_BUFFER

NOTE: XVBA compressed data buffers have a different data/bit layout for H.264, VC-1 and MPEG2. Please read the spec carefully and implement the appropriate buffer for every codec in XVBA.

4.1 Picture Descriptor Buffer

XVBA compressed data type: XVBA_PICTURE_DESCRIPTOR_BUFFER

Picture descriptor buffer contains information on picture size, structure, postprocessing, decode references, chroma format, etc. This buffer is the same for all supported codecs.

```
typedef struct
{
       //VC-1, MPEG2 bitstream references
      void               *past_surface;
      void               *future_surface;

      unsigned int       profile;
      unsigned int       level;

      unsigned int       width_in_mb;
      unsigned int       height_in_mb;
      unsigned int       picture_structure;

      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       XVBA_avc_sps_reserved                 : 26;
               } avc;

               struct {
                        unsigned int       postprocflag                : 1;
                        unsigned int       pulldown                    : 1;
                        unsigned int       interlace                   : 1;
                        unsigned int       tfcntrflag                  : 1;
                        unsigned int       finterpflag                 : 1;
                        unsigned int       reserved                    : 1;
                        unsigned int       psf                         : 1;
                        unsigned int       second_field                : 1;
                        unsigned int       XVBA_vc1_sps_reserved       : 24;
               } vc1;

               unsigned int       flags;
      } sps_info;


      unsigned char      chroma_format;
      unsigned char      avc_bit_depth_luma_minus8;
      unsigned char      avc_bit_depth_chroma_minus8;
      unsigned char      avc_log2_max_frame_num_minus4;
```

```
        unsigned char       avc_pic_order_cnt_type;
        unsigned char       avc_log2_max_pic_order_cnt_lsb_minus4;
        unsigned char       avc_num_ref_frames;
        unsigned char       avc_reserved_8bit;

        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        XVBA_avc_pps_reserved                 : 23;
                } avc;

                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        XVBA_vc1_pps_reserved       : 4;
                } vc1;

                unsigned int        flags;
        } pps_info;


        unsigned char       avc_num_slice_groups_minus1;
        unsigned char       avc_slice_group_map_type;
        unsigned char       avc_num_ref_idx_l0_active_minus1;
        unsigned char       avc_num_ref_idx_l1_active_minus1;

        char                avc_pic_init_qp_minus26;
        char                avc_pic_init_qs_minus26;
        char                avc_chroma_qp_index_offset;
        char                avc_second_chroma_qp_index_offset;

        unsigned short      avc_slice_group_change_rate_minus1;
        unsigned short      avc_reserved_16bit;

        unsigned int        avc_frame_num;
        unsigned int        avc_frame_num_list[16];      // bit 31 is used to indicate long/short term
        int                 avc_curr_field_order_cnt_list[2];
        int                 avc_field_order_cnt_list[16][2];

        unsigned char       avc_slice_group_map[810];

        int                 avc_intra_flag;
        int                 avc_reference;

        int                 XVBA_reserved[14];

    } XVBAPictureDescriptor;
```

4.1.1 Common fields of `XVBAPictureDescriptor` structure

The fields below are common for all supported codes:

profile

The field specifies a subset of algorithmic features and limits that shall be supported by all decoders conforming to that profile.
All H264 profile types are specified in H264/AVC1 specification.
All VC1 profile types are specified in VC1 specification.
AMD UVD hardware acceleration supports the following profiles
H264:

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)

VC1:

4 = Simple profile (for the VC1 field *PROFILE* = 0)
5 = Main profile (for the VC1 field *PROFILE* = 1)
6 = Advanced profile (for the VC1 field *PROFILE* = 3)

level

The field specifies restrictions on bitstreams and hence limits on the capabilities needed to decode the bitstreams. Levels are specified within each profile.

width_in_mb

The field specifies the width of each decoded picture in units of macroblocks.

height_in_mb

The field specifies the width of each decoded picture in units of macroblocks.

picture_structure

The field specifies the type of picture:

0 = Top field
1 = Bottom field
3 = Frame

chroma_format

The field 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 shall be inferred to be equal to 1 (4:2:0 chroma format).

avc_frame_num

The field is used as an identifier for pictures.
In H264, it is represented by *log2_max_frame_num_minus4* + 4 bits in the bitstream.

avc_intra_flag

The field specifies the prediction mode type in a frame/field.

1 = the flag specifies that 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 picture may be coded in Inter prediction mode.

avc_reference

The field specifies whether this picture is used as the reference picture.

1 = this picture is a reference

0 = non-reference picture

4.1.2 H264 specific fields of `XVBAPictureDescriptor` structure

avc_bit_depth_luma_minus8

The field corresponds to *bit_depth_luma_minus8* field in H263/AVC1 specification.
It specifies the bit depth of the samples of the luma array and the value of the luma quantisation parameter range offset,

BitDepthY = 8 + bit_depth_luma_minus8
QpBdOffsetY = 6 * bit_depth_luma_minus8

When *bit_depth_luma_minus8* is not present, it shall be inferred to be equal to 0.
avc_bit_depth_luma_minus8 shall be in the range of 0 to 4, inclusive.

avc_bit_depth_chroma_minus8

The field corresponds to *bit_depth_chroma_minus8* field in H263/AVC1 specification.
It specifies the bit depth of the samples of the chroma arrays and the value of the chroma quantisation parameter range offset, as specified by

BitDepthC = 8 + bit_depth_chroma_minus8
QpBdOffsetC = 6 * (bit_depth_chroma_minus8 + residual_colour_transform_flag)

When *avc_bit_depth_chroma_minus8* is not present, it shall be inferred to be equal to 0.
avc_bit_depth_chroma_minus8 shall be in the range of 0 to 4, inclusive.

avc_log2_max_frame_num_minus4

The field corresponds to *log2_max_frame_num_minus4* field in H263/AVC1 specification.
It 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 of *avc_log2_max_frame_num_minus4* shall be in the range of 0 to 12, inclusive.

avc_pic_order_cnt_type

The field corresponds to *pic_order_cnt_type* field in H263/AVC1 specification.
It specifies the method to decode picture order count (POC).
The value of *avc_pic_order_cnt_type* shall be in the range of 0 to 2, inclusive.

avc_log2_max_pic_order_cnt_lsb_minus4

The field corresponds to *log2_max_pic_order_cnt_lsb_minus4* field in H263/AVC1 specification.
It specifies the value of the variable *MaxPicOrderCntLsb* that is used
in the decoding process for picture order count as specified in subclause 8.2.1 as follows:

$$MaxPicOrderCntLsb = 2^{(log2\_max\_pic\_order\_cnt\_lsb\_minus4 + 4)}$$

The value of *avc_log2_max_pic_order_cnt_lsb_minus4* shall be in the range of 0 to 12, inclusive.

avc_num_ref_frames

The field corresponds to *num_ref_frames* field in H263/AVC1 specification.
It specifies the maximum number of short-term and long-term reference frames,
complementary reference field pairs, and non-paired reference fields that may be used
by the decoding process for inter prediction of any picture in the sequence.
num_ref_frames also determines the size of the sliding window operation.
The value of *avc_num_ref_frames* shall be in the range of 0 to *MaxDpbSize*, inclusive.

avc_reserved_8bit

It is the reserved field. It must be 0.

avc_num_slice_groups_minus1

The field corresponds to *num_slice_groups_minus1* field in H263/AVC1 specification.
It specifies the number of slice groups for a picture minus 1.
When *avc_num_slice_groups_minus1* is equal to 0, all slices of the picture belong to the same slice group.

0 = for H264 main and high profiles

0-7 = for H264 baseline profile

avc_slice_group_map_type

The field corresponds to *slice_group_map_type* field in H263/AVC1 specification.
It specifies how the mapping of slice group map units to slice groups is coded.
The value of *avc_slice_group_map_type* shall be in the range of 0 to 6, inclusive.

avc_num_ref_idx_l0_active_minus1

The field corresponds to *num_ref_idx_l0_active_minus1* field in H263/AVC1 specification.
It specifies the maximum reference index for reference picture list 0 that shall be used to decode each slice of the picture in which list 0 prediction is used when *num_ref_idx_active_override_flag* is equal to 0 for the slice. When *MbaffFrameFlag* is equal to 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 of *avc_num_ref_idx_l0_active_minus1* shall be in the range of 0 to 31, inclusive.

avc_num_ref_idx_l1_active_minus1

The field corresponds to *num_ref_idx_l1_active_minus1* field in H263/AVC1 specification.
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.

avc_pic_init_qp_minus26

The field corresponds to *pic_init_qp_minus26* field in H263/AVC1 specification.
It 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, and is modified further when a non-zero value of *mb_qp_delta* is decoded at the macroblock layer.
The value of *avc_pic_init_qp_minus26* shall be in the range of (26 + QpBdOffsetY) to +25, inclusive.

avc_pic_init_qs_minus26

The field corresponds to *pic_init_qs_minus26* field in H263/AVC1 specification.
It 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 of *avc_pic_init_qs_minus26* shall be in the range of -26 to +25, inclusive.

avc_chroma_qp_index_offset

The field corresponds to *chroma_qp_index_offset* field in H263/AVC1 specification.
It specifies the offset that shall be added to QPY and QSY for addressing the table of QPC values for the Cb chroma component.
The value of *avc_chroma_qp_index_offset* shall be in the range of -12 to +12, inclusive.

avc_second_chroma_qp_index_offset

The field corresponds to *second_chroma_qp_index_offset* field in H263/AVC1 specification.
It specifies the offset that shall be added to QPY and QSY for addressing the table of QPC values for the Cr chroma component.
The value of *avc_second_chroma_qp_index_offset* shall be in the range of 12 to +12, inclusive.
When *avc_second_chroma_qp_index_offset* is not present, it shall be inferred to be equal to *avc_chroma_qp_index_offset*.

avc_slice_group_change_rate_minus1

The field corresponds to slice_*group_change_rate_minus1* field in H263/AVC1 specification.
It is used to specify the variable *SliceGroupChangeRate*.
SliceGroupChangeRate specifies 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 of slice_group_change_rate_minus1 shall be in the range of 0 to PicSizeInMapUnits – 1, inclusive.
The *SliceGroupChangeRate* variable is specified as follows:

SliceGroupChangeRate = slice_group_change_rate_minus1 + 1

avc_reserved_16bit

Reserved field. It must be 0.

avc_frame_num_list[16]

It is not used for now. It must be 0.

avc_curr_field_order_cnt_list[2]

curr_field_order_cnt_list[0] corresponds to *TopFieldOrderCnt* in H264/ACV1 specification.
curr_field_order_cnt_list[1] corresponds to *BottomFieldOrderCnt* in H264/ACV1 specification.
The fields are used to determine initial picture orderings 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.

avc_field_order_cnt_list[16][2]

It is not used for now. It must be 0.

avc_slice_group_map[810]

It is not used for now. It must be 0.

4.1.2.1 The H264 picture parameters are defined in **sps_info** structure:

delta_pic_order_always_zero_flag

The field corresponds to the same field in H264/AVC1 specification.

1 = specifies that *delta_pic_order_cnt[0]* and *delta_pic_order_cnt[1]* are not present in the slice headers of the sequence and shall be inferred to be equal to 0.

0 = specifies that *delta_pic_order_cnt[0]* is present in the slice headers of the sequence and *delta_pic_order_cnt[1]* may be present in the slice headers of the sequence.

(See H264/ACV1 specification for reference)

gaps_in_frame_num_value_allowed_flag

The field corresponds to the same field in H264/AVC1 specification.
It specifies the allowed values of *frame_num* and the decoding process in case of an inferred gap between values of *frame_num*. (See H264/ACV1 specification for reference)

residual_colour_transform_flag

It corresponds to the same field in H264/AVC1 specification.

1 = specifies that the residual color transform is applied.

0 = specifies that the residual color transform is not applied.

When *residual_colour_transform_flag* is not present, it shall be inferred to be equal to 0.
(See H264/ACV1 specification for reference)

frame_mbs_only_flag

The field corresponds to the same field in H264/AVC1 specification.

1 = specifies that every coded picture of the coded video sequence is a coded frame containing only frame macroblocks.

0 = specifies that coded pictures of the coded video sequence may either be coded fields or coded frames.

(See H264/ACV1 specification for reference)

mb_adaptive_frame_field_flag

The field corresponds to the same field in H264/AVC1 specification.

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 *mb_adaptive_frame_field_flag* is not present, it shall be inferred to be equal to 0.

(See H264/ACV1 specification for reference)

direct_8x8_inference_flag

The field corresponds to the same field in H264/AVC1 specification.
It 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* shall be equal to 1.
(See H264/ACV1 specification for reference)

XVBA_avc_sps_reserved

It is the reserved field. It must be 0.

4.1.2.2 The H264 picture parameters are defined in **pps_info** structure:

entropy_coding_mode_flag

The field corresponds to the same field in H263/AVC1 specification.
It selects the entropy decoding method.

0 = Exp-Golomb coded or CAVLC
1 = CABAC

pic_order_present_flag

The field corresponds to the same field in H263/AVC1 specification.

1 = specifies that the picture order count related syntax elements are present in the slice headers.
0 = specifies that the picture order count related syntax elements are not present in the slice headers.

weighted_pred_flag

The field corresponds to the same field in H263/AVC1 specification.

1 = specifies that weighted prediction shall be applied to P and SP slices.
0 = specifies that weighted prediction shall not be applied to P and SP slices.

weighted_bipred_idc

The field corresponds to the same field in H263/AVC1 specification.

0 = the default weighted prediction shall be applied to B slices.
1 = explicit weighted prediction shall be applied to B slices.
2 = implicit weighted prediction shall be applied to B slices.

The value of *weighted_bipred_idc* shall be in the range of 0 to 2, inclusive.

deblocking_filter_control_present_flag

The field corresponds to the same field in H263/AVC1 specification.

1 = specifies that a set of syntax elements controlling the characteristics of the deblocking filter is present in the slice header.
0 = specifies that the set of syntax elements controlling the characteristics of the deblocking filter is not present in the slice headers and their inferred values are in effect.

constrained_intra_pred_flag

The field corresponds to the same field in H263/AVC1 specification.

1 = specifies constrained intra prediction, in which case prediction of macroblocks coded using Intra macroblock prediction modes only uses residual data decoded samples from I or SI macroblock types.
0 = specifies that intra prediction allows usage of residual data and decoded samples of neighboring macroblocks coded using Inter macroblock prediction modes for the prediction of macroblocks coded using Intra macroblock prediction modes.

redundant_pic_cnt_present_flag

The field corresponds to the same field in H263/AVC1 specification.

1 = that the redundant_pic_cnt syntax element is present 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 = specifies that the redundant_pic_cnt syntax element is not present in 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.

transform_8x8_mode_flag

The field corresponds to the same field in H263/AVC1 specification.

1 = specifies that the 8x8 transform decoding process may be in use.

0 = specifies that the 8x8 transform decoding process is not in use.

When transform_8x8_mode_flag is not present, it shall be inferred to be 0.

XVBA_avc_pps_reserved

It is the reserved field. It must be 0.

4.1.3 VC1 specific fields of `XVBAPictureDescriptor` structure

4.1.3.1 The VC1 picture parameters are defined in **sps_info** structure:

postprocflag

The field corresponds to *POSTPROC* field in VC1 specification.

It is a flag that indicates whether syntax element *POSTPROC* is present in picture headers.

pulldown

The field corresponds to *PULLDOWN* field in VC1 specification.

It is a flag that indicates whether the syntax elements *RPTFRM*, or *TFF* and *RFF* are present in picture headers.

interlace

The field corresponds to *INTERLACE* field in VC1 specification.

The individual frames may be coded using the progressive or interlace syntax when *INTERLACE* = 1.

If *INTERLACE* = 0, pictures are coded as single frames using the progressive syntax.

tfcntrflag

The field corresponds to *TFCNTRFLAG* field in VC1 specification.

It is a frame counter flag.

1 = indicates that the syntax element *TFCNTR* shall be present in the advanced profile picture headers.

0 = indicates that *TFCNTR* shall not be present in the picture header.

finterpflag

The field corresponds to *TFCNTRFLAG* field in VC1 specification.

It is 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.

reserved

The field corresponds to *RESERVED* field in VC1 specification.

It is the Reserved Advanced Profile Flag . It shall be set to 1.

It is used for bitstream control.

psf

The field corresponds to *PSF* field in VC1 specification.
It specifies the video source.

1 = the video source was Progressive Segmented Frame (PsF), and the display process should treat the decoded frames (field-pairs) as progressive.
0 = the display process may treat the decoded frames (field-pairs) according to the value of the *INTERLACE* syntax element.

second_field

The 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.

XVBA_vc1_sps_reserved

It is the reserved field. It must be 0.

4.1.3.2 The VC1 picture parameters are defined in **pps_info** structure:

panscan_flag

The field corresponds to the same field in VC1 specification.

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

The field corresponds to the same field in VC1 specification.
It is a Reference Frame Distance Flag.

1 = specifies that REFDIST syntax element is present in II, IP, PI or PP field picture headers.
0 = the REFDIST syntax element is not present.

loopfilter

The field corresponds to the same field in VC1 specification.

1 = specifies that loop filtering is enabled.
0 = specifies that loop filtering is not enabled.

If the stream PROFILE is Simple profile, the LOOPFILTER shall have the value 0.

fastuvmc

The field corresponds to the same field in VC1 specification.
It is 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 is 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 profile, the *FASTUVMC* shall have the value 0.

extended_mv

The field corresponds to the same field in VC1 specification.
It is the Extended Motion Vector Flag.
It specifies whether extended motion vectors are enabled (value 1) or disabled (value 0).

This bit shall always set to 0 for the Simple Profile.
For the Main Profile, the extended motion vector mode shall indicate the possibility of extended motion vectors in P and B pictures.

dquant

The field corresponds to the same field in VC1 specification.
It specifies whether or not the quantization step size may 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* shall be 0.
In the Main profile, if *MULTIRES* = 1, *DQUANT* shall be 0.

vstransform

The field corresponds to the same field in VC1 specification.
The specifies whether variable-sized transform coding is enabled for the sequence.

1 = variable-sized transform coding shall be enabled.
0 = variable-sized transform coding shall not be enabled.

overlap

The field corresponds to the same field in VC1 specification.
It specifies whether Overlapped Transforms are used.

1 = Overlapped Transforms may be used.
0 = Overlapped Transforms is not used.

quantizer

The field corresponds to the same field in VC1 specification.
It 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

The field corresponds to the same field in VC1 specification.

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.

maxbframes

The field corresponds to the same field in VC1 specification.
It specifies the maximum number of consecutive B frames between I or P frames.

0 = there are no B frames in the sequence.
0-7 = the number of B Frames may be present in the sequence.

rangered

The field corresponds to the same field in VC1 specification.
specifies whether range reduction is used for each frame.

1 = there shall be a syntax element in each frame header (*RANGEREDFRM*) that indicates whether range reduction is used for that frame.
0 = the syntax element *RANGEREDFRM* is not present, and range reduction shall not used.

RANGERED shall be set to zero in Simple profile.

syncmarker

The field corresponds to the same field in VC1 specification.
It indicates whether synchronization markers may be present in the bitstream.
This bit shall always be set to 0 in the simple profile.
In the main profile, the synchronizations markers may be present if *SYNCMARKER* =1,
and the markers shall not be present if *SYNCMARKER* = 0.

multires

The field corresponds to the same field in VC1 specification.
It is a Multiresolution Coding flag which specifies whether the frames may be coded at smaller resolutions than the specified frame resolution.
Resolution changes shall only be allowed on I pictures.

1 = the frame level *RESPIC* syntax element shall be present which indicates the resolution for that frame.
0 = *RESPIC* shall not be present.

reserved

The field corresponds to *Reserved6* field in VC1 specification.
It shall be set to 1 and other values shall be forbidden.
It is used to control a video stream.

range_mapy_flag

The field corresponds to the same field in VC1 specification.
The Range Mapping Luma Flag specifies whether RANGE_MAPY is present in within the entry header.

1 = RANGE_MAPY is present in within the entry header
0 = RANGE_MAPY is not present in within the entry header

range_mapy

The field corresponds to the same field in VC1 specification.
The Range Mapping Luma value shell be present if *range_mapy_flag* is set to 1.
The value of *range_mapy* shall 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 shall be scaled according to the formula:

Y[n] = CLIP ((((Y[n] – 128) * (RANGE_MAPY + 9) + 4) >> 3) + 128);

range_mapuv_flag

The field corresponds to the same field in VC1 specification.
The Range Mapping Color-Difference Flag specifies whether *RANGE_MAPUV* is present in within 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 field corresponds to the same field in VC1 specification.
The Range Mapping Color-Difference value shell be present if range_mapy_flag is set to 1.
The value of *range_mapuv* shall 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 shall 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);

XVBA_vc1_pps_reserved

It is the reserved field. It must be 0.

4.2 Data Buffer

4.2.1 Bitstream decode (H.264 and VC-1)

XVBA compressed data type: XVBA_DATA_BUFFER

Data size/packing description of XVBA_DATA_BUFFER is in the XVBA_DATA_CTRL_BUFFER. XVBA_DATA_BUFFER must be 128 byte aligned.

4.2.2 MPEG2 iDCT level decode

XVBA compressed data type: XVBA_DATA_BUFFER

This buffer stores residual data in the following format:

```
typedef struct
{
  struct
  {
        unsigned short  index: 15; //contains rates scan index of the coefficient within the block.
                                   //cannot be greater or equal to (block width * block height)
        unsigned short  endofblock:  1;
  } idx;
  short           coeff;             //value of the coefficient in the block; mismatch control and
                                     // clipping is host's responsibility

} XVBAMpeg2Residual;
```

Notes:

- If 'endofblock' is 1, it indicates that the current coefficient is last one in the current block.
- XVBA mpeg2 blocks are in arbitrary ordering.

4.3 Data Control Buffer or Slice Buffer

4.3.1 Bitstream decode (H.264 and VC-1)

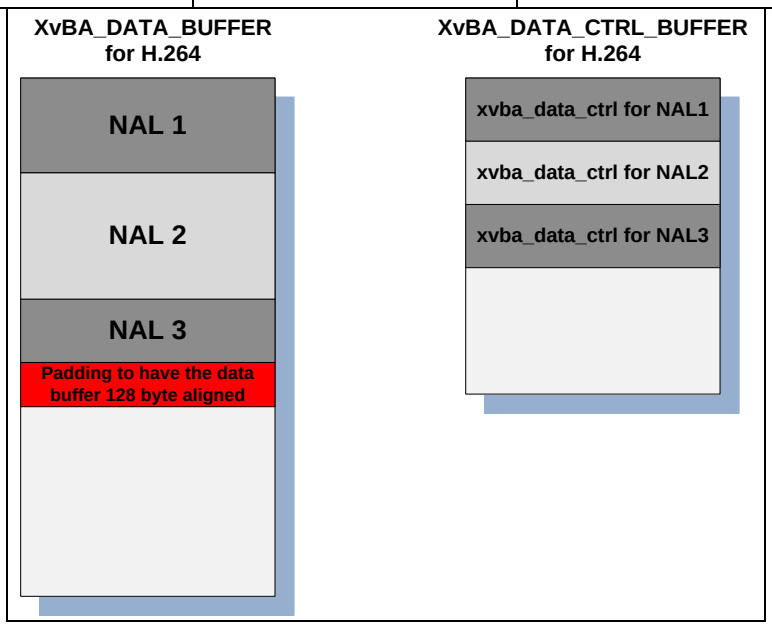
XVBA compressed data type: XVBA_DATA_CTRL_BUFFER

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

} XVBA_data_ctrl;
```

4.3.1.1 Data Control Buffer and Data Buffer Relation for H.264 and VC-1

XVBA_DATA_BUFFER contains blocks of compressed bitstream data. Decoder (host) stores the data blocks size/location information in XVBA_DATA_CTRL_BUFFER. Every data block has its own XVBA_data_ctrl data structure. XVBA_DATA_BUFFER must be 128 byte aligned.



XVBA_DATA_BUFFER and XVBA_DATA_CTRL_BUFFER example for h.264

4.3.2 MPEG2 iDCT level decode

XVBA compressed data type: XVBA_DATA_CTRL_BUFFER

This buffer stores macroblock information for the mpeg2 stream. There are 2 different structures: one for the I and the other for the non-I macroblocks. Host builds XVBA data control buffer usint appropriate structure for every macroblock: Intra macroblocks shall use `XVBA_mpeg2_intra_mb`, where non-Intra macroblocks (P and B) shall use `XVBA_mpeg2_nonintra_mb`.

```
// define for motion_type
#define XVBA_PREDICTION_FIELD         0x01
#define XVBA_PREDICTION_FRAME         0x02
#define XVBA_PREDICTION_DUAL_PRIME    0x03
#define XVBA_PREDICTION_16x8          0x02
#define XVBA_SECOND_FIELD             0x00000004
```

Motion vectors:

```
typedef struct
{
  short horizontal;
  short vertical;
} XVBAMpeg2MV;
```

Intra MB:

```
typedef struct
{
  unsigned short      mb_address;
  struct
  {
    unsigned short    mb_intra              : 1;
    unsigned short    motion_fw             : 1;
    unsigned short    motion_back           : 1;
    unsigned short    reserved2             : 2;
    unsigned short    field_residual        : 1;
    unsigned short    mb_scan_mode          : 2;
    unsigned short    motion_type           : 2;
```

```
    unsigned short     reserved1              : 2;
    unsigned short     motion_vector_sel0     : 1;
    unsigned short     motion_vector_sel1     : 1;
    unsigned short     motion_vector_sel2     : 1;
    unsigned short     motion_vector_sel3     : 1;
  } mpeg2data1;

  struct
  {
    unsigned int       mb_data_resid_location : 24;
    unsigned int       skipped_mb             : 8;
  } mpeg2data2;

  unsigned short       pattern_code;
  unsigned char        numcoeff[6];

} XVBAMpeg2IntraMB;
```

Non-Intra MB:

```
typedef struct
{
  unsigned short       mb_address;
  struct
  {
    unsigned short     mb_intra               : 1;
    unsigned short     motion_fw              : 1;
    unsigned short     motion_back            : 1;
    unsigned short     reserved2              : 2;
    unsigned short     field_residual         : 1;
    unsigned short     mb_scan_mode           : 2;
    unsigned short     motion_type            : 2;
    unsigned short     reserved1              : 2;
    unsigned short     motion_vector_sel0     : 1;
    unsigned short     motion_vector_sel1     : 1;
    unsigned short     motion_vector_sel2     : 1;
    unsigned short     motion_vector_sel3     : 1;
  } mpeg2data1;

  struct
  {
    unsigned int       mb_data_resid_location : 24;
    unsigned int       skipped_mb             : 8;
  } mpeg2data2;

  unsigned short       pattern_code;
  unsigned char        numcoeff[6];

  XVBAMpeg2MV          motion_vector[4];

} XVBAMpeg2NonIntraMB;
```

4.4 QM Buffer

4.4.1 Bitstream decode (H.264 and VC-1)

XVBA compressed data type: XVBA_QM_BUFFER

Used for H.264 only in XVBA version 1. XVBA_QM_BUFFER contains quantization matrix data.

```
typedef struct
{
  unsigned char        bScalingLists4x4[6][16];
  unsigned char        bScalingLists8x8[2][64];

} XVBAQuantMatrixAVC;
```

4.4.2 MPEG2 iDCT level decode

Not used for XVBA MPEG2 iDCT level decoding.