Advanced Micro Devices

Advanced Media Framework – Video Decoder

Programming Guide

Disclaimer

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information.

Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

AMD, the AMD Arrow logo, ATI Radeon™, CrossFireX™, LiquidVR™, TrueAudio™ and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Windows™, Visual Studio and DirectX are trademark of Microsoft Corp.

Copyright Notice

Notice Regarding Standards. AMD does not provide a license or sublicense to any Intellectual Property Rights relating to any standards, including but not limited to any audio and/or video codec technologies such as MPEG-2, MPEG-4; AVC/H.264; HEVC/H.265; AAC decode/FFMPEG; AAC encode/FFMPEG; VC-1; and MP3 (collectively, the "Media Technologies"). For clarity, you will pay any royalties due for such third party technologies, which may include the Media Technologies that are owed as a result of AMD providing the Software to you.

MIT license

Contents

1. Introduction
2. AMF Video Decoder Component
    - 2.1 Component Initialization
    - 2.2 Configuring the Decoder
    - 2.3 Submitting Input and Retrieving Output
    - 2.4 Terminating the Decoder Component
3. Sample Applications

1 Introduction

This document provides a complete description of the AMD Advanced Media Framework (AMF) Video Decoder Component. This component exposes the AMD Video Decoder, which provides hardware accelerated video decoding functionality for the following codecs:

- MPEG2
- MPEG4
- WMV3
- VC1
- H.264 (including SVC and MVC)
- Motion JPEG
- H.265 (HEVC)
- VP9
- AV1

2 AMF Video Decoder Component

Video Decoder accepts elementary streams of the above codecs as input and produces output in a sequence of DirectX 9 surfaces, DirectX 11.1 textures, Vulkan surfaces or textures, or DirectX 12 surfaces or textures.

The AMF Video Decoder component does not deal with multimedia container formats or demultiplexing of video, audio and other streams. The application using AMF Video Decoder must deal with these tasks on its own.

Include `public/include/components/VideoDecoderUVD.h`.

2.1 Component Initialization

The AMF Video Decoder Component should be initialized using the following sequence:

1. Create an AMF Context and initialize it for one of the following:
    i. DirectX 12
    ii. DirectX 11.1
    iii. DirectX 9
    iv. Vulkan
    v. OpenGL
    vi. OpenCL
2. Determine the codec and create an instance of the AMF Video Decoder object using the `AMFFactory::CreateComponent` method passing the above `AMFContext` interface as parameter. Use the following component IDs depending on the codec selected:

Component ID	Description
AMFVideoDecoderUVD_MPEG2	MPEG-2
AMFVideoDecoderUVD_MPEG4	MPEG-4 including MPEG-4 part 2
AMFVideoDecoderUVD_WMV3	WMV3
AMFVideoDecoderUVD_VC1	VC1
AMFVideoDecoderUVD_H264_AVC	h.264 AVC
AMFVideoDecoderUVD_H264_MVC	h.264 MVC (multi-stream)
AMFVideoDecoderUVD_H264_SVC	h.264 SVC (scalable video codec)
AMFVideoDecoderUVD_MJPEG	Motion JPEG
AMFVideoDecoderHW_H265_HEVC	h.265/HEVC (8-bit 4:2:0 sampling)
AMFVideoDecoderHW_H265_MAIN10	h.265/HEVC with Main 10 profile (8- or 10-bit 4:2:0 sampling)
AMFVideoDecoderHW_VP9	VP9 – (8-bit 4:2:0 sampling)
AMFVideoDecoderHW_VP9_10BIT	VP9 – (10-bit 4:2:0 sampling)
AMFVideoDecoderHW_AV1	AV1

Table 1. AMF components IDs depending on codec selection

3. Configure the decoder component by setting the necessary properties using the `AMFPropertyStorage::SetProperty` method on the decoder object.
4. Call the `AMFComponent::Init` method of the decoder object. The `format` parameter must be set to `AMF_SURFACE_NV12` for all codecs. The Motion JPEG codec supports the `AMF_SURFACE_YUY2` format in addition to `AMF_SURFACE_NV12`.

2.2 Configuring the Decoder

AMF Decoder can be configured using the following properties that need to be set before initialization:

Name	Type
AMF_VIDEO_DECODER_SURFACE_COPY	amf_bool
AMF_VIDEO_DECODER_EXTRADATA	AMFBufferPtr
AMF_VIDEO_DECODER_FRAME_RATE	amf_double
AMF_TIMESTAMP_MODE	amf_int64
AMF_VIDEO_DECODER_ADAPTIVE_RESOLUTION_CHANGE	amf_bool
AMF_VIDEO_DECODER_REORDER_MODE	amf_int64
AMF_VIDEO_DECODER_DPB_SIZE	amf_int64
AMF_VIDEO_DECODER_ENABLE_SMART_ACCESS_VIDEO	amf_bool
AMF_VIDEO_DECODER_SKIP_TRANSFER_SMART_ACCESS_VIDEO	amf_bool

Table 2. AMF Video Decoder pre-initialization properties

On supported APU + GPU systems, there is an opportunity to use SmartAccess Video. SmartAccess Video - an optimization logic which enables the parallelization of encode and decode streams across multiple Video Codec Engine (VCN) hardware instances – empowers apps to process streams faster through seamless job distribution across available hardware. With a simple enablement of the encoder and decoder control flags, the SmartAccess Video logic will optimally use hardware resources to

benefit media apps. Follow the `SMART_ACCESS_VIDEO` tag in the documentation to search for the property flags to set. On systems without SmartAccess Video support, the `SMART_ACCESS_VIDEO` properties have no effect.

Name: `AMF_VIDEO_DECODER_SURFACE_COPY`

Values: `true`, `false`

Default Value: `false`

Description: Output samples are copied to newly allocated `AMFSurface` objects. This reduces decoder performance, but avoids the `AMF_DECODER_NO_FREE_SURFACES` error. Enable when the rest of the pipeline is significantly slower than the rate of submission of input samples.

Name: `AMF_VIDEO_DECODER_EXTRADATA`

Values: `AMFBufferPtr`

Default Value: `NULL`

Description: Set SPS/PPS on the output stream. The property contains a pointer to an `AMFBuffer` object containing the data.

Name: `AMF_VIDEO_DECODER_FRAME_RATE`

Values: `0` to `DBL_MAX`

Default Value: `0.0`

Description: The frame rate in FPS.

Name: `AMF_TIMESTAMP_MODE`

Values: `AMF_TIMESTAMP_MODE_ENUM`: `AMF_TS_PRESENTATION`, `AMF_TS_SORT`, `AMF_TS_DECODE`

Default Value: `AMF_TS_PRESENTATION`

Description:

- `AMF_TS_PRESENTATION` – timestamps are generated based on the set frame rate (default). Use of this mode is necessary when decoding elementary streams with no timestamps on input frames. This is the most reliable option.
- `AMF_TS_SORT` – timestamps are transferred from input samples to output samples and then sorted to ensure that timestamps on output frames appear in ascending order.
- `AMF_TS_DECODE` – timestamps are transferred from input samples to output samples. No sorting is performed.

Name: `AMF_VIDEO_DECODER_ADAPTIVE_RESOLUTION_CHANGE`

Values: `true`, `false`

Default Value: `false`

Description: When set to `false` (default) output surfaces will be reallocated on re-initialization when input resolution changes. When set to `true`, output surfaces will be reused if the new resolution is lower.

Name: `AMF_VIDEO_DECODER_REORDER_MODE`

Values: `AMF_VIDEO_DECODER_MODE_ENUM` : `AMF_VIDEO_DECODER_MODE_REGULAR` , `AMF_VIDEO_DECODER_MODE_COMPLIANT` , `AMF_VIDEO_DECODER_MODE_LOW_LATENCY`

Default Value: `AMF_VIDEO_DECODER_MODE_REGULAR`

Description: Determines frame reordering policy, which defines the decoder latency, i.e. the number of frames to be submitted before output becomes available:

- `AMF_VIDEO_DECODER_MODE_REGULAR` – number of reference frames+1.
- `AMF_VIDEO_DECODER_MODE_COMPLIANT` – based on the profile, up to `16` frames.
- `AMF_VIDEO_DECODER_MODE_LOW_LATENCY` – low latency mode, output becomes available immediately. The decoder expects a stream with no frame reordering. B- and P-frames are allowed as long as they do not cause frame reordering.

Name: `AMF_VIDEO_DECODER_DPB_SIZE`

Values: `0` ... `32`

Default Value: `1`

Description: The minimum required number of surfaces for frame reordering.

Name: `AMF_VIDEO_DECODER_ENABLE_SMART_ACCESS_VIDEO`

Values: `true` , `false`

Default Value: `false`

Description: When set to `true` , enables the SmartAccess Video feature, which optimally allocates the decoding task on supported APU/GPU pairings.

Name: `AMF_VIDEO_DECODER_SKIP_TRANSFER_SMART_ACCESS_VIDEO`

Values: `true` , `false`

Default Value: `false`

Description: When set to `false` and the SmartAccess Video feature is enabled, decoded frames are transferred back to the initial APU/GPU. When this is set to `true` , surfaces remain on the decoder device, which may not be the same as the initial APU/GPU.

The following read-only properties can be read to obtain information about the current stream, as well as decoder capabilities:

Name	Type
AMF_VIDEO_DECODER_ALLOC_SIZE	AMFSize
AMF_VIDEO_DECODER_CURRENT_SIZE	AMFSize
AMF_VIDEO_DECODER_CAP_NUM_OF_STREAMS	amf_int64

Table 3. AMF Video Decoder read-only properties

Name: `AMF_VIDEO_DECODER_ALLOC_SIZE`

Values: `(0, 0)` ... `(2147483647, 2147483647)`

Default Value: `(1920,1088)`

Description: Allocated output surface size.

Name: `AMF_VIDEO_DECODER_CURRENT_SIZE`

Values: `(0, 0)` ... `(2147483647, 2147483647)`

Default Value: `(0,0)`

Description: Current resolution.

Name: `AMF_VIDEO_DECODER_CAP_NUM_OF_STREAMS`

Values: `MAX_STREAM_NUMBER`

Default Value: `16`

Description: Retrieved through the `AMFCaps` interface, the maximum number of streams the decoder can support simultaneously. This property is deprecated.

2.3 Submitting Input and Retrieving Output

Once the Decoder component is successfully initialized, you may start submitting input samples to it. Input samples must be submitted as `AMFBuffer` objects.

At the same time poll for output by calling `AMFComponent::QueryOutput` on the Decoder object. Polling for output samples can be done either from the same thread or from another thread.

Suspend submission of input samples when `AMFComponent::SubmitInput` returns `AMF_INPUT_FULL` or `AMF_DECODER_NO_FREE_SURFACES`. Continue to poll for output samples and process them as they become available.

When `AMF_REPEAT` is returned from `AMFComponent::SubmitInput()` this means that the currently submitted buffer has more than one frame and needs another `AMFComponent::SubmitInput()` call to process the remaining data before getting any new data. This second `AMFComponent::SubmitInput()` should be invoked with `NULL` as the argument to perform the processing of the remaining data.

2.4 Terminating the Decoder Component

To terminate the Decoder component, call the `Terminate` method, or simply destroy the object. Ensure that the context used to create the Decoder component still exists during termination.

3 Sample Applications

A sample application demonstrating the use of the Decoder component in AMF is available as part of the AMF SDK in `public/samples/CPPSample/SimpleDecoder`. The sample takes a file with an h.264, an h.265 or an .ivf elementary stream and decodes it to a file containing uncompressed raw frames.

To run the sample, execute the `SimpleDecoder.exe <input file name>` command at the command prompt. Note that the output file can be large, ensure there's sufficient disk space available.