rocDecode documentation

Release 0.6.0

Advanced Micro Devices, Inc.

Aug 19, 2024

CONTENTS

rocDecode provides APIs, utilities, and samples that you can use to easily access the video decoding features of your media engines (VCNs). It also allows interoperability with other compute engines on the GPU using Video Acceleration API (VA-API)/HIP. To learn more, see *What is rocDecode?*

The code is open and hosted at https://github.com/ROCm/rocDecode.

Install

- *Quick start*
- *rocDecode installation*

The documentation is structured as follows:

Tutorials

- GitHub samples

How to

- *Using rocDecode*

Conceptual

- *Video decoding pipeline*

Reference

- *API library*
- *Functions*
- *Data structures*

To contribute to the documentation, refer to Contributing to ROCm.

You can find licensing information on the Licensing page.

CHAPTER

ONE

WHAT IS ROCDECODE?

AMD GPUs contain one or more media engines (VCNs) that provide fully accelerated, hardware-based video decoding. Hardware decoders consume lower power than CPU-based decoders. Dedicated hardware decoders offload decoding tasks from the CPU, boosting overall decoding throughput. With proper power management, decoding on hardware decoders can lower the overall system power consumption and improve decoding performance.

Using the rocDecode API, you can decode compressed video streams while keeping the resulting YUV frames in video memory. With decoded frames in video memory, you can run video post-processing using ROCm HIP, thereby avoiding unnecessary data copies via PCIe bus. You can post-process video frames using scaling or color conversion and augmentation kernels (on a GPU or host) in a format for GPU/CPU-accelerated inferencing and training.

In addition, you can use the rocDecode API to create multiple instances of video decoder based on the number of available VCNs in a GPU device. By configuring the decoder for a device, all available VCNs can be used seamlessly for decoding a batch of video streams in parallel.

For more information, refer to the *Video decoding pipeline*.

CHAPTER

TWO

ROCDECODE QUICK START INSTALLATION

To install the rocDecode runtime with minimum requirements, follow these steps:

1. Install core ROCm components (ROCm 6.1.0 or later) using the native package manager installation instructions.
   - Register repositories
   - Register kernel-mode driver
   - Register ROCm packages
   - Install kernel driver (`amdgpu-dkms`)–only required on bare metal install. Docker runtime uses the base `dkms` package irrespective of the version installed.

2. Install rocDecode runtime package. rocDecode only provides the `librocdecode.so` library (the runtime package only installs the required core dependencies).

Ubuntu

```
sudo apt install rocdecode
```

RHEL

```
sudo yum install rocdecode
```

SLES

```
sudo zypper install rocdecode
```

CHAPTER

THREE

INSTALLATION

rocDecode SDK is a high-performance video decode SDK for AMD GPUs. Using the rocDecode API, you can access the video decoding features available on your GPU.

3.1 Tested configurations

- Linux
  - Ubuntu: 20.04/22.04
  - RHEL: 8/9
- ROCm
  - rocm-core: 6.1.0.60100-28
  - amdgpu-core: 1:6.1.60100-1731559
- libva-dev: 2.7.0-2/2.14.0-1
- mesa-amdgpu-va-drivers: 1:24.1.0
- mesa-amdgpu-dri-drivers - *24.1.0.60200*
- FFmpeg: 4.2.7/4.4.2-0
- rocDecode Setup Script: V2.0.0

3.2 Supported codecs

H.265 (HEVC) - 8 bit, and 10 bit

3.3 Prerequisites

- Linux distribution
  - Ubuntu: 20.04/22.04
  - RHEL: 8/9
- ROCm-supported hardware (`gfx908` or higher is required)
- Install ROCm 6.1.0 or later with amdgpu-install

  - Run: `--usecase=rocm`
  - To install rocDecode with minimum requirements, follow the *quick start instructions*

- Video Acceleration API Version `1.5.0` or later - `Libva` is an implementation for VA-API

```
sudo apt install libva-dev
```

- AMD VA drivers

```
sudo apt install mesa-amdgpu-va-drivers
```

- CMake 3.5 or later

```
sudo apt install cmake
```

- pkg-config

```
sudo apt install pkg-config
```

- FFmpeg runtime and headers - for tests and samples

```
sudo apt install ffmpeg libavcodec-dev libavformat-dev libavutil-dev
```

Note:

- All package installs are shown with the `apt` package manager. Use the appropriate package manager for your operating system.
- On `Ubuntu 22.04` - Additional package required: `libstdc++-12-dev`

```
sudo apt install libstdc++-12-dev
```

- On `RHEL` / `SLES` - Additional packages required: `libdrm-amdgpu` and `mesa-amdgpu-dri-drivers`

```
sudo apt install libdrm-amdgpu mesa-amdgpu-dri-drivers
```

3.3.1 Prerequisites setup script

For your convenience, we provide the setup script, rocDecode-setup.py, which installs all required dependencies. Run this script only once.

```
python rocDecode-setup.py  --rocm_path [ ROCm Installation Path - optional (default:/opt/
↪rocm)]
                           --developer [ Setup Developer Options - optional (default:ON)␣
↪[options:ON/OFF]]
```

3.4 Installation instructions

To install rocDecode, you can use *Package install* or *Source install*.

3.4.1 Package install

To install rocDecode runtime, development, and test packages, run the line of code for your operating system.

Ubuntu

```
sudo apt install rocdecode rocdecode-dev rocdecode-test
```

RHEL

```
sudo yum install rocdecode rocdecode-devel rocdecode-test
```

SLES

```
sudo zypper install rocdecode rocdecode-devel rocdecode-test
```

Note: Package install auto installs all dependencies.

- Runtime package: `rocdecode` only provides the rocdecode library `librocdecode.so`
- Development package: `rocdecode-dev``or ``rocdecode-devel` provides the library, header files, and samples
- Test package: `rocdecode-test` provides CTest to verify installation

3.4.2 Source install

To build rocDecode from source, run:

```
git clone https://github.com/ROCm/rocDecode.git
cd rocDecode
mkdir build && cd build
cmake ../
make -j8
sudo make install
```

Run tests (this requires FFmpeg dev install):

```
make test
```

To run tests with verbose option, use `make test ARGS="-VV"`.

Make package:

```
sudo make package
```

3.5 Verify installation

The installer copies:

- Libraries into `/opt/rocm/lib`
- Header files into `/opt/rocm/include/rocdecode`
- Samples folder into `/opt/rocm/share/rocdecode`
- Documents folder into `/opt/rocm/share/doc/rocdecode`

To verify your installation using a sample application, run:

```
mkdir rocdecode-sample && cd rocdecode-sample
cmake /opt/rocm/share/rocdecode/samples/videoDecode/
make -j8
./videodecode -i /opt/rocm/share/rocdecode/video/AMD_driving_virtual_20-H265.mp4
```

To verify your installation using the `rocdecode-test` package, run:

```
mkdir rocdecode-test && cd rocdecode-test
cmake /opt/rocm/share/rocdecode/test/
ctest -VV
```

This test package installs the CTest module.

3.6 Samples

You can access samples to decode your videos in our GitHub repository. Refer to the individual folders to build and run the samples.

FFmpeg is required for sample applications and `make test`. To install FFmpeg, refer to the instructions listed for your operating system:

Ubuntu

```
sudo apt install ffmpeg libavcodec-dev libavformat-dev libavutil-dev
```

RHEL

Install FFmpeg development packages manually or use the rocDecode-setup.py script

SLES

Install FFmpeg development packages manually or use the rocDecode-setup.py script

3.7 Docker

You can find rocDecode Docker containers in our GitHub repository.

3.8 Documentation

Run the following code to build our documentation locally.

```
cd docs
pip3 install -r sphinx/requirements.txt
python3 -m sphinx -T -E -b html -d _build/doctrees -D language=en . _build/html
```

For more information on documentation builds, refer to the Building documentation page.

3.9 Hardware capabilities

The following table shows the codec support and capabilities of the VCN for each supported GPU architecture.

GPU Architecture	VCN Generation	Number of VCNs	H.265/HI	Max width, Max height - H.265	H.264/A	Max width, Max height - H.264
gfx908 - MI1xx	VCN 2.5.0	2	Yes	4096, 2176	Yes	4096, 2160
gfx90a - MI2xx	VCN 2.6.0	2	Yes	4096, 2176	Yes	4096, 2160
gfx940, gfx942 - MI3xx	VCN 3.0	3	Yes	7680, 4320	Yes	4096, 2176
gfx941 - MI3xx	VCN 3.0	4	Yes	7680, 4320	Yes	4096, 2176
gfx1030, gfx1031, gfx1032 - Navi2x	VCN 3.x	2	Yes	7680, 4320	Yes	4096, 2176
gfx1100, gfx1102 - Navi3x	VCN 4.0	2	Yes	7680, 4320	Yes	4096, 2176
gfx1101 - Navi3x	VCN 4.0	1	Yes	7680, 4320	Yes	4096, 2176

CHAPTER

FOUR

USING ROCDECODE

To learn how to use the rocDecode SDK library and its different utilities, follow these instructions:

4.1 1. API overview

All rocDecode APIs are exposed in the header files `rocdecode.h` and `rocparser.h`. You can find these files in the *api* folder in the rocDecode repository.

The samples use the `RocVideoDecoder` user class provided in `roc_video_dec.h` in the `utils` folder of the rocDecode repository.

A video parser (defined in `rocparser.h`) is needed to extract and decode headers from the bitstream in order to organize the data into a structured format for the hardware decoder. The parser is critical in video decoding, as it controls the decoding and display of a bitstream's individual frames and fields.

The parser object in `rocparser.h` has three main APIs:

- `rocDecCreateVideoParser()`
- `rocDecParseVideoData()`
- `rocDecDestroyVideoParser()`

4.2 2. Create a parser object

The `rocDecCreateVideoParser()` API creates a video parser object for the codec that you specify. The API takes `max_num_decode_surfaces`, which determines the Decoded Picture Buffer (DPB) size for decoding. When creating a parser object, the application must register certain callback functions with the driver, which is called from the parser during decode.

- `pfn_sequence_callback` is called when the parser encounters a new sequence header. The parser informs you of the minimum number of surfaces needed by the parser's DPB to successfully decode the bitstream. In addition, the caller can set additional parameters, like `max_display_delay`, to control frame decoding and display.
- The `pfn_decode_picture` callback function is triggered when a picture is set for decoding.
- The `pfn_display_picture` callback function is triggered when a frame in display order is ready to be consumed by the caller.
- The `pfn_get_sei_msg` callback function is triggered when your Supplementation Enhancement Information (SEI) message is parsed and sent back to the caller.

4.3 3. Parse video data

Elementary stream video packets extracted from the de-multiplexer are fed into the parser using the `rocDecParseVideoData()` API.

During this call, the parser triggers the callbacks as it encounters a new sequence header, receives compressed frame/field data ready to be decoded, or when it's ready to display a frame. If any of the callbacks return a failure, it is propagated back to the application so the decoding can be ended gracefully.

4.4 4. Query decode capabilities

The `rocDecGetDecoderCaps()` API allows you to query the capabilities of the underlying hardware video decoder. Decoder capabilities usually include supported codecs, maximum resolution, and bit-depth.

The following pseudo-code illustrates the use of this API. The application handles the error appropriately for non-supported decoder capabilities.

```
RocdecDecodeCaps decode_caps;
memset(&decode_caps, 0, sizeof(decode_caps));
decode_caps.codec_type = p_video_format->codec;
decode_caps.chroma_format = p_video_format->chroma_format;
decode_caps.bit_depth_minus_8 = p_video_format->bit_depth_luma_minus8;

ROCDEC_API_CALL(rocDecGetDecoderCaps(&decode_caps));

if(!decode_caps.is_supported) {
    ROCDEC_THROW("Rocdec:: Codec not supported on this GPU: ", ROCDEC_NOT_SUPPORTED);
    return 0;
}

if ((p_video_format->coded_width > decode_caps.max_width) ||
    (p_video_format->coded_height > decode_caps.max_height)) {

    std::ostringstream errorString;
    errorString << std::endl
                << "Resolution          : " << p_video_format->coded_width << "x" << p_
↪video_format->coded_height << std::endl
                << "Max Supported (wxh) : " << decode_caps.max_width << "x" << decode_
↪caps.max_height << std::endl
                << "Resolution not supported on this GPU ";

    const std::string cErr = errorString.str();
    ROCDEC_THROW(cErr, ROCDEC_NOT_SUPPORTED);
    return 0;
}
```

4.5 5. Create a decoder

`rocDecCreateDecoder()` creates an instance of the hardware video decoder object and provides you with a handle upon successful creation. Refer to the `RocDecoderCreateInfo` structure for information about the parameters passed for creating the decoder. For example, `RocDecoderCreateInfo::codec_type` represents the codec type of the video. The decoder handle returned by `rocDecCreateDecoder()` must be retained for the entire decode session because the handle is passed along with the other decoding APIs. In addition, you can inform display or crop dimensions along with this API.

4.6 6. Decode the frame

After de-multiplexing and parsing, you can decode bitstream data containing a frame/field using hardware.

Use the `rocDecDecodeFrame()` API to submit a new frame for hardware decoding. Underneath the driver, the Video Acceleration API (VA-API) is used to submit compressed picture data to the driver. The parser extracts all the necessary information from the bitstream and fills the `RocdecPicParams` structure that's appropriate for the codec. The high-level `RocVideoDecoder` class connects the parser and decoder used for all sample applications.

The `rocDecDecodeFrame()` call takes the decoder handle and the pointer to the `RocdecPicParams` structure and initiates the video decoding using VA-API.

4.7 7. Query the decoding status

After submitting a frame for decoding, you can call `rocDecGetDecodeStatus()` to query the decoding status for a given frame. A structure pointer, `RocdecDecodeStatus*`, is filled and returned.

The API inputs are:

- `decoder_handle`: A `RocDecoder` handler, `rocDecDecoderHandle`.
- `pic_idx`: An *int* value for the `picIdx` for which you want a status in order to index of the picture.
- `decode_status`: A pointer to `RocdecDecodeStatus` as a return value.

The API returns one of the following statuses:

- Invalid (0): Decode status is not valid.
- In Progress (1): Decoding is in progress.
- Success (2): Decoding was successful and no errors were returned.
- Error (8): The frame was corrupted, but the error was not concealed.
- Error Concealed (9): The frame was corrupted and the error was concealed.
- Displaying (10): Decode is complete, display in progress.

4.8 8. Prepare the decoded frame for further processing

The decoded frames can be used for further postprocessing using `rocDecGetVideoFrame()`. The successful completion of `rocDecGetVideoFrame()` indicates that the decoding process is complete and the device memory pointer is inter-opped into the ROCm HIP address space in order to further process the decoded frame in device memory. The caller gets the necessary information on the output surface, such as YUV format, dimensions, and pitch from this call. In the high-level `RocVideoDecoder` class, we provide four different surface type modes for the mapped surface, as specified in `OutputSurfaceMemoryType`.

```
typedef enum OutputSurfaceMemoryType_enum {
    OUT_SURFACE_MEM_DEV_INTERNAL = 0,       /**<  Internal interopped decoded surface␣
↪memory **/
    OUT_SURFACE_MEM_DEV_COPIED = 1,         /**<  decoded output will be copied to a␣
↪separate device memory **/
    OUT_SURFACE_MEM_HOST_COPIED = 2         /**<  decoded output will be copied to a␣
↪separate host memory **/
    OUT_SURFACE_MEM_NOT_MAPPED = 3          /**<  decoded output is not available␣
↪(interop won't be used): useful for decode only performance app*/
} OutputSurfaceMemoryType;
```

If the mapped surface type is `OUT_SURFACE_MEM_DEV_INTERNAL`, the direct pointer to the decoded surface is provided. You must call `ReleaseFrame()` (`RocVideoDecoder` class). If the requested surface type is `OUT_SURFACE_MEM_DEV_COPIED` or `OUT_SURFACE_MEM_HOST_COPIED`, the internal decoded frame is copied to another buffer, either in device memory or host memory. After that, it's immediately unmapped for re-use by the `RocVideoDecoder` class.

Refer to the `RocVideoDecoder` class and samples for details on how to use these APIs.

4.9 9. Reconfigure the decoder

You can call `rocDecReconfigureDecoder()` to reuse a single decoder for multiple clips or when the video resolution changes during the decode. The API currently supports resolution changes, resize parameter changes, and target area parameter changes for the same codec without destroying an ongoing decoder instance. This can improve performance and reduce overall latency.

The API inputs are:

- `decoder_handle`: A `RocDecoder` handler, `rocDecDecoderHandle`.
- `reconfig_params`: You must specify the parameters for the changes in `RocdecReconfigureDecoderInfo`. The width and height used for reconfiguration cannot exceed the values set for `max_width` and `max_height`, defined in `RocDecoderCreateInfo`. If you need to change these values, you have to destroy and recreate the session.

Note: You must call `rocDecReconfigureDecoder()` during `RocdecParserParams::pfn_sequence_callback`.

4.10 10. Destroy the decoder

You must call the `rocDecDestroyDecoder()` to destroy the session and free up resources.

The API input is:

- `decoder_handle`: A `RocDecoder` handler, `rocDecDecoderHandle`.

The API returns a `RocdecDecodeStatus` value.

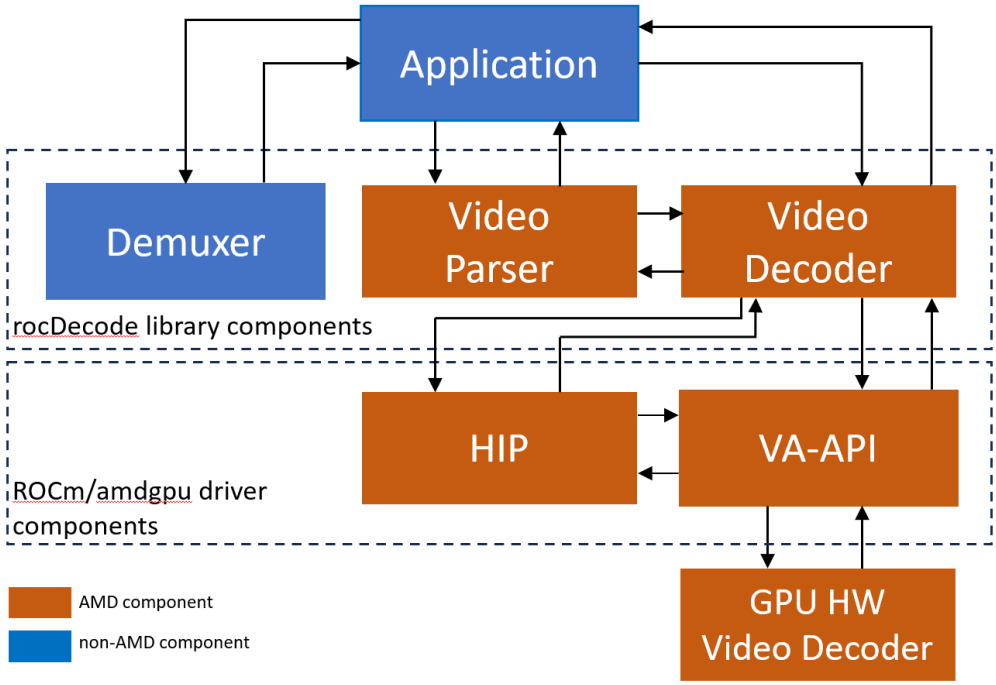
4.11 11. Destroy the parser

You must call `rocDecDestroyVideoParser()` to destroy the parser object and free up all allocated resources at the end of video decoding.

CHAPTER

FIVE

VIDEO DECODING PIPELINE



There are three main components in rocDecode:

- Demuxer: Our demuxer is based on FFmpeg, a leading multimedia framework. For more information, refer to the FFmpeg website.
- Video parser APIs
- Video decoder APIs

rocDecode follows this workflow:

1. The demuxer extracts a segment of video data and sends it to the video parser.
2. The video parser extracts crucial information, such as picture and slice parameters, and sends it to the decoder APIs.
3. The hardware receives the picture and slice parameters, then decodes a frame using Video Acceleration API (VA-API).
4. This process repeats in a loop until all frames have been decoded.

Steps in decoding video content for applications (available in the rocDecode Toolkit):

1. Demultiplex the content into elementary stream packets (FFmpeg)
2. Parse the demultiplexed packets into video frames for the decoder provided by rocDecode API.
3. Decode compressed video frames into YUV frames using rocDecode API.
4. Wait for the decoding to finish.
5. Get the decoded YUV frame from amd-gpu context to HIP (using VAAPI-HIP interoperability under ROCm).
6. Run HIP kernels in the mapped YUV frame. For example, format conversion, scaling, object detection, classification, and others.
7. Release the decoded frame.

Note: YUV is a color space that represents images using luminance (Y) for brightness and two chrominance components (U and V) for color information.

The preceding steps are demonstrated in the sample applications located in our GitHub repository directory.

CHAPTER

SIX

FILE LIST

CHAPTER

SEVEN

GLOBALS

CHAPTER

EIGHT

DATA STRUCTURES

CHAPTER

NINE

LICENSE

MIT License

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.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.