JH7110 Multimedia Developing Guide

VisionFive 2

Version: 1.0

Date: 2022/12/30

Doc ID: JH7110-DGEN-013

Legal Statements

Important legal notice before reading this documentation.

PROPRIETARY NOTICE

Contact Us

Address: Room 502, Building 2, No. 61 Shengxia Rd., China (Shanghai) Pilot Free Trade Zone, Shanghai, 201203, China

Website: http://www.starfivetech.com

Email:

- Sales: sales@starfivetech.com
- Support: support@starfivetech.com

Contents

List of Tables

List of Figures

Preface

About this guide and technical support information.

About this document

This document mainly provides the SDK developers with the programing basics and debugging know-how for the multimedia module of the StarFive next generation SoC platform - JH7110.

Audience

This document mainly serves the multimedia module relevant driver developers. If you are developing other modules, place a request to your sales or support consultant for our complete documentation set on JH7110.

Revision History

Table 0-1 Revision History

Version	Released	Revision
1.0	2022/12/30	First public release.

Notes and notices

The following notes and notices might appear in this guide:

- **Tip:**
  Suggests how to apply the information in a topic or step.
- **Note:**
  Explains a special case or expands on an important point.
- **Important:**
  Points out critical information concerning a topic or step.
- **CAUTION:**
  Indicates that an action or step can cause loss of data, security problems, or performance issues.
- **Warning:**
  Indicates that an action or step can result in physical harm or cause damage to hardware.

1. Introduction

1.1. Function Introduction

JH7110 SoC Platform supports the following features and specifications on the multimedia module.

- FFmpeg and common plug-ins
- GStreamer and common plug-ins
- OpenMax API for hard encoding and decoding

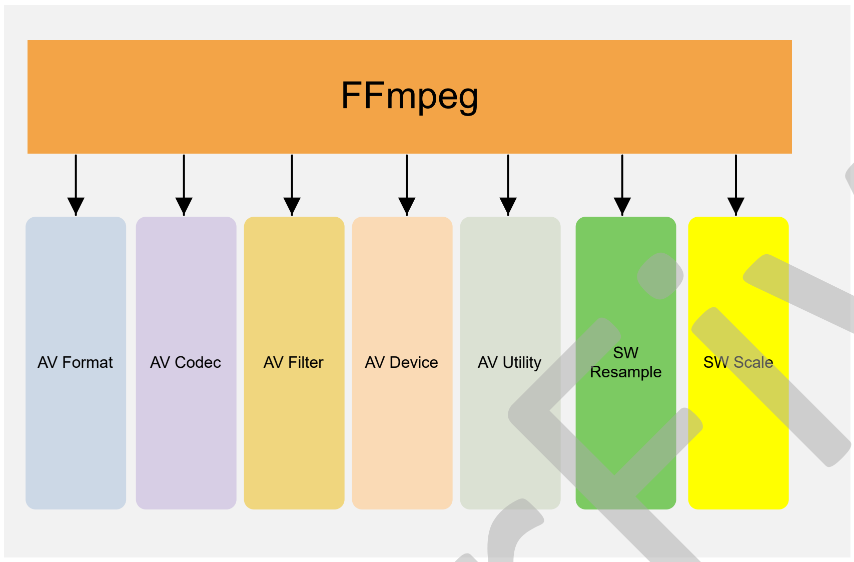
2. FFmpeg

FFmpeg is the leading multimedia framework, able to decode, encode, transcode, mux, demux, stream, filter and play pretty much anything that humans and machines have created.

FFmpeg is widely used for format transcoding, basic editing (trimming and concatenation), video scaling, video post-production effects and standards compliance.

The following figure shows the FFmpeg framework.

Figure 2-1 FFmpeg Framework



2.1. Source Code Location

Locate the JH7110 *Software Development Kit (SDK)* with the following information.

- **Repository**: https://github.com/starfive-tech/VisionFive2
- **Branch**: JH7110_VisionFive2_devel
- **Tag**: VF2_v2.4.4

The source code of FFmpeg and its related plug-ins can be downloaded from the network, and then generated by adding the patches provided by StarFive. For details, please refer to https://github.com/starfive-tech/buildroot/tree/JH7110_VisionFive2_devel/package/ffmpeg.

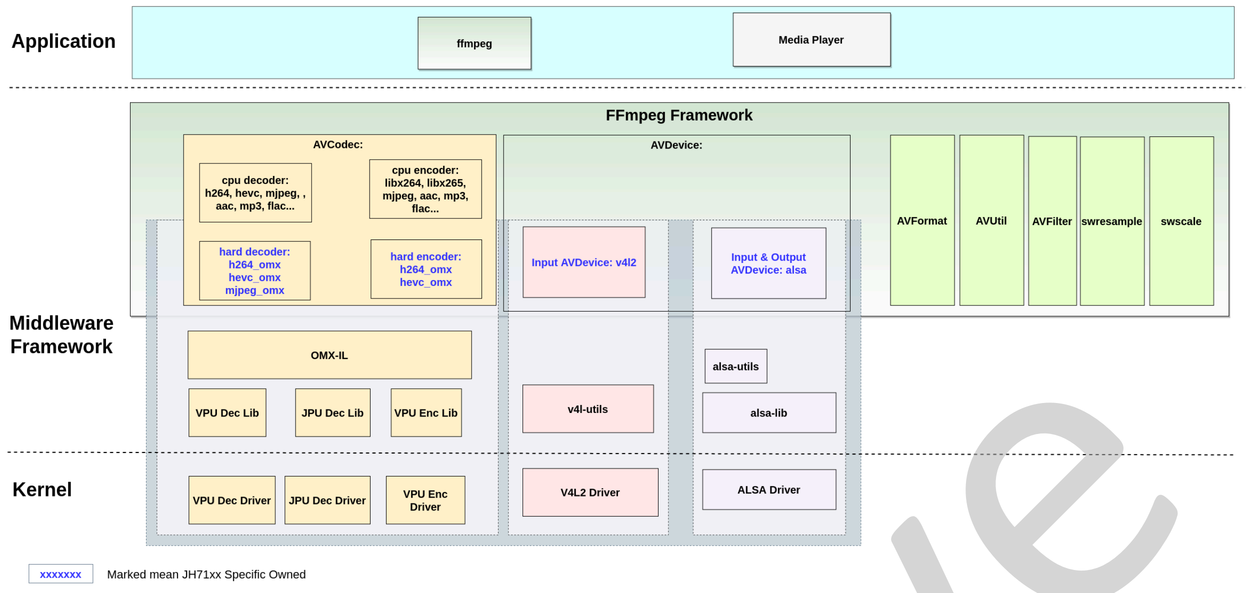
The system currently supports FFmpeg version 4.4.1.

The source code of FFmpeg and its required plug-ins are in the following directory: `buildroot/package/ffmpeg/`

2.2. Block Diagram

The following figure shows the block diagram of FFmpeg.

Figure 2-2 FFmpeg Block Diagram



2.3. Build FFmpeg

In the JH7110 software SDK, FFmpeg is built buildroot, you can download the source code by referring to Source Code Location *(on page 10)*.

But in Debian, FFmpeg is compiled independently, StarFive may provide a Debian installation binary, or enable users to run the following command for installation.

```
apt install ffmpeg
```

The following procedure introduces how to build FFmpeg in buildroot.

1. Enable the related macros (which are enabled by default).

   ```
   BR2_PACKAGE_FFMPEG_GPL=y
   BR2_PACKAGE_FFMPEG_FFPROBE=y
   BR2_PACKAGE_FFMPEG_AVRESAMPLE=y
   BR2_PACKAGE_FFMPEG_POSTPROC=y
   BR2_PACKAGE_WAVE511=y
   BR2_PACKAGE_WAVE420L=y
   BR2_PACKAGE_CODAJ12=y
   BR2_PACKAGE_SF_OMX_IL=y
   ```

   The complete list of plug-ins can be found in **menuconfig > Target packages > Audio and video applications > ffmpeg** and **menuconfig > Target packages > starfive packages**.

2. Build them in the SDK root directory directly with the following commands.

   ```
   $ make buildroot_initramfs-menuconfig
   $ make buildroot_rootfs-menuconfig
   ```

2.4. General Commands

2.4.1. ffprobe

The command `ffprobe` is used to gather information from multimedia streams and prints it in a human- and machine-readable fashion. For example it can be used to check the format of the container used by a multimedia stream and the format and type of each media stream contained in it.

It can be used to inspect the media source like video file, audio file, support most container like MP4, AVI, and WAV, etc.

The following code block provides an example of executing the command.

```
ffprobe 1080p.mp4
```

2.4.2. ffmpeg

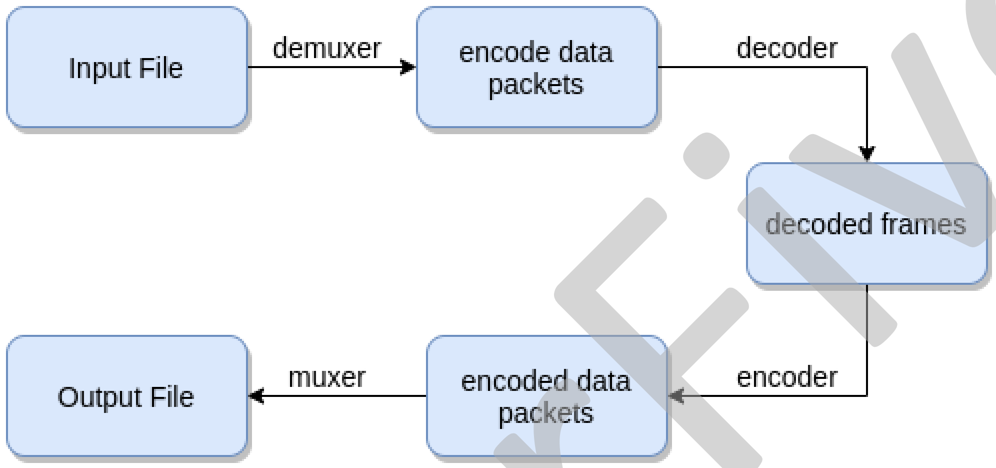
The command `ffmpeg` is used to call a very fast video and audio converter that can also grab from a live audio/video source. It can also convert between arbitrary sample rates and resize video on the fly with a high quality poly-phase filter.

The following code block provides an example of executing the command.

```
ffmpeg -i input.avi -b:v 64k -bufsize 64k -vcodec h264 output.avi
```

The above transcoding process in ffmpeg for each output can be described by the following diagram.

Figure 2-3 ffmpeg Transcoding Process



2.4.3. ffplay

The command `ffplay` is used to call a very simple and portable media player using the FFmpeg libraries and the SDL library. It is mostly used as a testbed for the various FFmpeg application interfaces.

The following code block provides an example of executing the command.

```
ffplay -i input.avi
```

Note:

Debian provides the `ffplay` command, but currently the command is not supported yet in the JH7110 software SDK version VF2_515_v2.4.x.

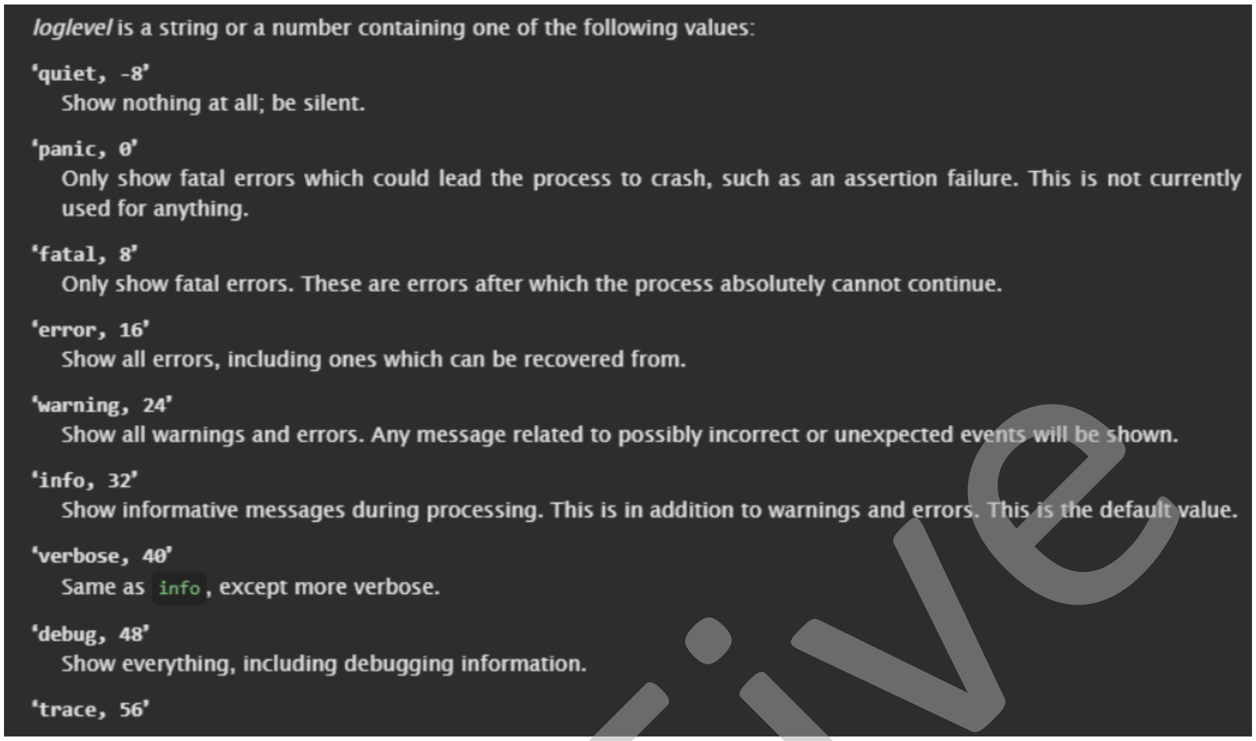
2.4.4. Enabling logs

To enable the log function for `ffmpeg` and `ffplay`, use the log level setting option:

```
ffmpeg -v [flags+]loglevel -i input output
```

The system provides the following log levels.

Figure 2-4 Log Level



loglevel is a string or a number containing one of the following values:

'quiet, -8'
Show nothing at all; be silent.

'panic, 0'
Only show fatal errors which could lead the process to crash, such as an assertion failure. This is not currently used for anything.

'fatal, 8'
Only show fatal errors. These are errors after which the process absolutely cannot continue.

'error, 16'
Show all errors, including ones which can be recovered from.

'warning, 24'
Show all warnings and errors. Any message related to possibly incorrect or unexpected events will be shown.

'info, 32'
Show informative messages during processing. This is in addition to warnings and errors. This is the default value.

'verbose, 40'
Same as `info`, except more verbose.

'debug, 48'
Show everything, including debugging information.

'trace, 56'

2.5. Common Libraries

2.5.1. Libavcodec

The library **libavcodec** provides a generic encoding/decoding framework and contains multiple decoders and encoders for audio, video and subtitle streams, and several bitstream filters.

The OMX plug-ins which support hard codec/encoder also be included in the **libavcodec** library.

JH7110 software SDK supports the following OMX plug-ins.

- h264_omx
- hevc_omx
- mjpeg_omx

2.5.1.1. h264_omx

The plug-in **h264_omx** supports H264 video hard decoding through VPU.

Note:
The plug-in does not support hard encoding.

The following code block shows the special options supported by the plug-ins.

Table 2-1 h264_omx Special Options

Option	Type	Explanation
`-omx_pix_fmt`	string	Set the decoding pixel format for **h264_omx** decoder. The following formats are supported: yuv420p, nv12, nv21. (default "yuv420p")

Table 2-1 h264_omx Special Options (continued)

Option	Type	Explanation
`-scale_width`	int	Set the scaling width for OMX (Only zoom out is supported, ceil 8 (width/8) and minimum 1/8) (from 0 to INT_MAX) (default 0)
`-scale_height`	int	Set the scaling height for OMX (Only zoom out is supported, ceil 8 (height/8) and minimum 1/8) (from 0 to INT_MAX) (default 0)

The following code block shows an example of decoding a 1920 × 1080 H.264 file output to NV21 1280 × 720 YUV.

```
ffmpeg  -vcodec h264_omx -omx_pix_fmt nv21 -scale_width 1280 -scale_height 720 -i 1080p_30FPS_AVC_420_omx.h264
 -frames 100 ff_720p_AVC_i420.yuv -hide_banner -y -v verbose
```

2.5.1.2. hevc_omx

The plug-in **hevc_omx** supports both H.265 video hard encoding and hard decoding through VPU.

The following code block shows the special options supported by the plug-ins.

Table 2-2 hevc_omx Special Options

Option	Type	Explanation
`-omx_pix_fmt`	string	Set the decoding pixel format for **hevc_omx** decoder. The following formats are supported: yuv420p, nv12, nv21. (default "yuv420p")
`-scale_width`	int	Set the scaling width for OMX (Only zoom out is supported, ceil 8 (width/8) and minimum 1/8) (from 0 to INT_MAX) (default 0)
`-scale_height`	int	Set the scaling height for OMX (Only zoom out is supported, ceil 8 (height/8) and minimum 1/8) (from 0 to INT_MAX) (default 0)

The following code block shows an example of decoding a 1920 × 1080 H.265 file output to NV21 1280 × 720 YUV.

```
ffmpeg  -vcodec hevc_omx -omx_pix_fmt nv21 -scale_width 1280 -scale_height 720 -i 1080p_30FPS_AVC_420_omx.h264
 -frames 100 ff_720p_AVC_i420.yuv -hide_banner -y -v verbose
```

2.5.1.3. mjpeg_omx

The plug-in **mjpeg_omx** supports JPEG picture and MJPEG video hard decoding through VPU.

Note:
The plug-in does not support hard encoding

The following code block shows the special options supported by the plug-ins.

Table 2-3 mjpeg_omx Special Options

Option	Type	Explanation
`-omx_pix_fmt`	string	Set the decoding pixel format for **mjpeg_omx** decoder. The following formats are supported: yuv420p, nv12, nv21, nv16, yuv422p, yuyv422, yvyu422, uyvy422, yu-v444p.
`-scale_width`	int	Set the scaling width for OMX (Only zoom out is supported, ceil 8 (width/8) and minimum 1/8) (from 0 to INT_MAX) (default 0)
`-scale_height`	int	Set the scaling height for OMX (Only zoom out is supported, ceil 8 (height/8) and minimum 1/8) (from 0 to INT_MAX) (default 0)

Table 2-3 mjpeg_omx Special Options (continued)

Option	Type	Explanation
`-mirror`	int	Mirror 0 (none), 1 (V), 2 (H) or 3 (VH), cannot be set at the same time as pixel format conversion, rotation and crop (from 0 to 3) (default 0)
`-rotation`	int	Rotation 0 (0), 1 (90), 2 (180) or 3 (270), cannot be set at the same time as pixel format conversion, mirror and crop (from 0 to 3) (default 0)
`-crop`	string	Crop <x>, <y>, <w>, <h>: crop coord and width/height(from left/top, must be an integer multiple of 16), cannot be set at the same time as pixel format conversion, mirror and rotation.

The following code block shows an example of decoding a 3840x2160 JPEG picture output to nv21 1920x1080 yuv.

```
ffmpeg -nostdin -nostats -c:v mjpeg_omx -omx_pix_fmt nv21 -scale_width 1 -scale_height 1 -i
 JPG_3840_2160_420.jpg ff_JPG_1920_2160_i420_scale.yuv -hide_banner -y -v verbose
```

2.5.2. Libavformat

The library **libavformat** provides a generic framework for multiplexing and demultiplexing (muxing and demuxing) audio, video and subtitle streams. It encompasses multiple muxers and demuxers for multimedia container formats.

It also supports several input and output protocols to access a media resource.

2.5.3. Libavdevice

The library **libavdevice** provides a generic framework for grabbing from and rendering to many common multimedia input/output devices, and supports several input and output devices, including V4L2(Video4Linux2), VfW, DShow, and ALSA.

2.5.4. Libavutil

The library **libavutil** is a utility library to aid portable multimedia programming. It contains safe portable string functions, random number generators, data structures, additional mathematics functions, cryptography and multimedia related functionality (like enumerations for pixel and sample formats). It is not a library for code needed by both **libavcodec** and **libavformat**.

2.5.5. Libswscale

The library **libswscale** performs highly optimized image scaling and color-space and pixel format conversion operations.

2.5.6. Libavfilter

The library **libavfilter** provides a generic audio/video filtering framework containing several filters, sources and sinks.

2.5.7. Libswresample

The library **libswresample** performs highly optimized audio re-sampling, re-matrixing and sample format conversion operations.

2.6. Use Examples

2.6.1. File Property Inspection

The following code block provides an example of inspecting properties of an audio file.

```
ffprobe test1-16k-32b.aac
```

The following code block provides an example of inspecting properties of a video file.

```
ffmpeg -i 1080p_h264_aac.mp4
```

2.6.2. Video Decoding

General Example

The section provides an examples for decoding the H.264 and H.265 video stream decoded via FFmpeg OMX plug-in.

The example requires the following plug-in(s).

- **h264_omx**
- **hevc_omx**

The following code block shows the example in general.

```
ffmpeg -vcodec hevc_omx -omx_pix_fmt yuv420p -i 1080p_30FPS_HEVC_420_omx.h265 -f rawvideo -pix_fmt yuv420p -y
 ff_1080p_30fps_HEVC_i420.yuv -hide_banner -y -v verbose
```

Scenarios

The following code block provides examples in different scenarios.

- Input with MP4 container:

```
ffmpeg  -vcodec h264_omx -i 1080p_h264_aac.mp4  -frames 100 -pix_fmt yuv420p 1080p_h264_aac.yuv
```

- Input with AVI container:

```
ffmpeg -vcodec h264_omx  -omx_pix_fmt nv21  -i 4K_60Fps_AVC_High15_2.avi -frames 70 -pix_fmt
 nv21 4K_60Fps_AVC_avi_NV21.yuv
```

- Output with NV12 format YUV:

```
ffmpeg  -vcodec h264_omx -omx_pix_fmt nv12 -i 4K_30FPS_AVC_420_omx.h264 -frames 70 -f rawvideo -pix_fmt
 nv12 -y ff_4K_AVC_NV12.yuv  -hide_banner -y -v verbose
```

- Output with scale to 1280x720 YUV:

```
ffmpeg  -vcodec h264_omx -omx_pix_fmt yuv420p -i 720p_30FPS_AVC_420_omx.h264 -frames 70 -f rawvideo
 -pix_fmt yuv420p -y ff_720p_AVC_i420.yuv -hide_banner -y -v verbose
```

- For H265 video decoding:

```
ffmpeg  -vcodec hevc_omx -i 1080p_h265_aac.mp4  -frames 100 -pix_fmt yuv420p 1080p_h265_aac.yuv
```

2.6.3. Video Encoding

General Example

The section provides an example for encoding the video streams (in YUV format) into H.265 formats via the FFmpeg OMX plug-in.

The example requires the following plug-in(s).

- **hevc_omx**

The following code block shows the example in general.

```
ffmpeg -s 1920*1080 -i 1080p_h264_aac.yuv -vcodec hevc_omx -frames 20 1080p_h264_aac.265
```

Scenarios

The following code block provides examples in different scenarios.

- Output with MP4 container:

```
ffmpeg -s 1920*1080 -i 1080p_h264_aac.yuv -vcodec hevc_omx -frames 20 1080p_h264_aac.mp4
```

- Output with AVI container:

```
ffmpeg -s 1920*1080 -pix_fmt yuv420p -i 1080P_30Fps_HEVC_High_I420.yuv -vcodec hevc_omx -b:v 5000k
 -vtag hev1 1080P_30Fps_HEVC.avi
```

- Output with MKV container:

```
ffmpeg -s 1920*1080 -pix_fmt yuv420p -i 1080P_30Fps_HEVC_High_I420.yuv -vcodec hevc_omx
 -b:v 5000k 1080P_30Fps_HEVC.mkv
```

- Output with MOV container:

```
ffmpeg -s 1920*1080 -pix_fmt yuv420p -i 1080P_30Fps_HEVC_High_I420.yuv -vcodec hevc_omx -b:v 5000k
 1080P_30Fps_HEVC.mov
```

- Encoding support NV12 format:

```
ffmpeg  -s 3840*2160 -r 30 -pix_fmt nv12 -i ff_4K_HEVC_NV12.yuv -vcodec hevc_omx -b:v 5000k -y
 ff_4K_HEVC_NV12_omx_30fps.h265 -hide_banner -y -v verbose
```

- Encoding support NV21 format:

```
ffmpeg  -s 3840*2160 -r 30 -pix_fmt nv21 -i ff_4K_HEVC_NV21.yuv -vcodec hevc_omx -b:v 5000k -y
 ff_4K_HEVC_NV21_omx_30fps.h265 -hide_banner -y -v verbose
```

- Encoding with bit rate 5 Mbps:

```
ffmpeg  -s 1920*1080  -pix_fmt yuv420p -i ff_1080P_HEVC_I420.yuv -vcodec hevc_omx -b:v 5000k -y
 ff_1080p_HEVC_NV12_omx_25fps.h265  -hide_banner -y -v verbose
```

- Encoding with bit rate 5 Mbps and frame rate 15 fps:

```
ffmpeg  -s 1920*1080  -r 15 -pix_fmt yuv420p -i ff_1080p_30fps_HEVC_i420.yuv -vcodec hevc_omx
 -b:v 5000k -y ff_1080p_HEVC_NV12_omx_15fps.h265 -hide_banner -y -v verbose
```

- Encoding with GOP30:

```
ffmpeg -s 1920*1080 -pix_fmt yuv420p -i 1080p_30fps_HEVC_i420.yuv -vcodec hevc_omx -g 30
 ff_1080p_HEVC_i420_omx_g30.h265
```

2.6.4. Video Transcoding

The following code block provides an example of transcoding and scale an H.264 1920 × 1080 file to H.265 1280 × 720.

```
ffmpeg -v verbose -vcodec h264_omx -scale_width 1280 -scale_height 720 -i /1080p_h264_aac.mp4 -vcodec hevc_omx
 -acodec copy /out/720p_h265_aac.mp4 -y
```

The following code block provides an example of transcoding and scale an H.264 + ACC MP4 file to MPEG4 + MP3 MP4 (through soft transcoding).

```
ffmpeg -i 1080p_h264_aac.mp4 -vcodec mpeg4 -acodec libmp3lame 1080p_mpeg4_mp3.mp4
```

2.6.5. JPEG Image Decoding

General Example

The section provides an example for decoding a JPEG image file to YUV 420p with no image size scaling.

The example requires the following plug-in(s).

- **mjpeg_omx**

The following code block shows the example in general.

```
ffmpeg -nostdin -nostats -c:v mjpeg_omx -omx_pix_fmt yuv420p -i JPG_3840_2160_420.jpg
 ff_JPG_3840_2160_i420_scale.yuv -hide_banner -y -v verbose
```

Scenarios

The following code block provides examples in different scenarios.

- Output YUV with NV12 format, scale width to 1/2, and scale height to 1/2:

```
ffmpeg -nostdin -nostats -c:v mjpeg_omx -omx_pix_fmt nv12 -scale_width 1 -scale_height 1 -i
 MJPEG_3840_2160_444.mjpeg ff_MJPEG_1920_1080_nv12_scale.yuv -hide_banner -y -v verbose
```

- Output YUV with NV16 format:

```
ffmpeg -c:v mjpeg_omx -omx_pix_fmt nv16 -i JPG_3840_2160_444.jpg ff_JPG_3840_2160_nv16.yuv -hide_banner
 -y -v verbose
```

- Output YUV with NV12 format and mirror horizontal:

```
ffmpeg -nostdin -nostats -c:v mjpeg_omx -omx_pix_fmt nv12 -mirror 2 -i MJPEG_3840_2160_444.mjpeg
 ff_MJPEG_3840_2160_nv12_scale.yuv -hide_banner -y -v verbose
```

- Output YUV with NV12 format and rotate 90 degrees:

```
ffmpeg -nostdin -nostats -c:v mjpeg_omx -omx_pix_fmt nv12 -rotation 1 -i MJPEG_3840_2160_444.mjpeg
 ff_MJPEG_2160_3840_nv12_scale.yuv -hide_banner -y -v verbose
```

- Output YUV with crop at x:800 y:800 width:800 height:800:

```
ffmpeg -c:v mjpeg_omx -crop 800,800,800,800 -i JPG_3840_2160_444.jpg ff_JPG_800_800_crop.yuv
 -hide_banner -y
```

Note:
The crop width and height must be a value of 16 multiples.

2.6.6. MPEG Video Decoding

General Example

The section provides an example for decoding an MPEG video file to YUV.

The example requires the following plug-in(s).

- **mjpeg_omx**

The following code block shows the example in general.

```
ffmpeg -nostdin -nostats -c:v mjpeg_omx -omx_pix_fmt nv12 -scale_width 1 -i MJPEG_3840_2160_444.mjpeg
 ff_MJPEG_1920_2160_420_scale.yuv -hide_banner -y -v verbose
```

Scenarios

The following code block provides examples in different scenarios.

- Output YUV and mirror vertical:

```
ffmpeg -c:v mjpeg_omx -mirror 1 -i sample_3840x2160.mjpeg ff_MJPEG_3840_2160_I420.yuv -hide_banner -y
```

- output YUV and rotation 90 degrees:

```
ffmpeg -c:v mjpeg_omx -rotation 1 -i sample_3840x2160.mjpeg ff_MJPEG_3840_2160_rotation.yuv
 -hide_banner -y
```

- Output YUV with crop at x:800 y:800 width:800 height:800:

```
ffmpeg -c:v mjpeg_omx -crop 800,800,800,800 -i sample_3840x2160.mjpeg ff_MJPEG_800_800_crop.yuv
 -hide_banner -y
```

Note:
The crop width and height must be a value of 16 multiples.

2.6.7. Video Capturing

The following code block provides an example of capturing a 1080p video from an USB camera or sensor camera.

```
ffmpeg -f v4l2 -t 10 -r 30 -s 1920*1080 -i /dev/video1 test_v4l2.yuv
```

The following code block provides an example of capturing a 640 × 480 video from an USB camera or sensor camera.

```
ffmpeg -f v4l2 -t 10 -r 30 -s 640*480 -i /dev/video1 test_v4l2.yuv
```

2.6.8. Video Capturing and Encoding

The topic provides examples for capturing a video and encoding it to be an H.265 file.

The examples require the following plug-in(s).

- **hevc_omx**

The following code block shows the example in different scenarios.

- Capture from USB camera and encoding:

```
ffmpeg -f v4l2 -t 10 -r 30 -s 1920*1080 -i /dev/video4 -vcodec hevc_omx -vb 5000k -pix_fmt yuv420p
 ff_uvc_h265_1080p_420_30.h265
```

- Capture from IMX219 sensor and encoding:

```
ffmpeg -f v4l2 -t 20 -r 30 -s 1920*1080 -i /dev/video1 -vcodec hevc_omx -vb 5000k ff_219_1080p_30.h265
```

2.6.9. Audio Capturing

The topic provides examples for capturing an audio file.

The following code block shows the example in different scenarios.

- Capture audio from **wm8960**:

```
ffmpeg -f alsa -acodec pcm_s32le -i hw:0,1 output-alsa-wm8960.wav
```

- Capture audio from **wm8960** with 16 K sample rate and 2 channels, 32-bit depth:

```
ffmpeg -f alsa -i hw:0,0 -ar 16000 -ac 2 -acodec pcm_s32le ff_wm8960_16k_32b_c2.wav
```

- Capture audio from **ac108**:

```
ffmpeg -f alsa -acodec pcm_s32le -i hw:0,1 output-alsa-ac108.wav
```

- Capture audio from **ac108** with 16 K sample rate and 2 channels, 32-bit depth:

```
ffmpeg -f alsa  -ar 16000 -ac 2 -acodec pcm_s32le -i hw:0,0 ff_ac108_16k_32b_c2.wav
```

- Capture audio from TDM:

```
ffmpeg -f alsa -ar 32000 -ac 2 -acodec pcm_s32le -i hw:0,0 ff_tdm_32k_16b_c2.wav
```

- Capture audio from PDM:

```
ffmpeg -f alsa -i hw:0,0 -ar 16000 -ac 2 -acodec pcm_s32le ff_pdm_16k_32b_c2.wav
```

2.6.10. Audio and Video Capturing

The topic provides an example for capturing audio and video contents and encoding them into a MP4 file.

```
ffmpeg -f alsa -acodec pcm_s32le -i hw:0,1 -f v4l2 -s 640*480 -t 20 -r 30 -i /dev/video1 -vcodec h264_omx
 output-h264_alsa.mp4
```

2.6.11. Audio Playing

The topic provides examples for playing an audio file.

The following code block shows the example in different scenarios.

- Play an audio file with PWMDAC:

  Required devices: JH7110 EVB or VisionFive 2

```
ffmpeg -i test1-32k-16b.wav -f alsa -acodec pcm_s16le hw:0,1
```

- Play an audio with **wm8960**:

  Required devices: JH7110 EVB + **wm8960** audio board or VisionFive 2 + **wm8960** audio board

```
ffmpeg -i test1-16k-32b.wav -f alsa -acodec pcm_s32le hw:0,1
```

2.6.12. Audio Soft Decoding

The topic provides examples for soft decoding an audio file.

The following code block shows the example in different scenarios.

- Soft-decode a WAV file to audio RAW PCM:

```
ffmpeg -i test1-16k-32b.wav -f s32le -acodec pcm_s32le -ar 16000 -ac 2 output-test1-16k-32b.pcm
```

- Soft-decode a MP3 file to audio RAW PCM:

```
ffmpeg -i test1-16k-32b.mp3 -f s32le -acodec pcm_s32le -ar 16000 -ac 2 output-test1-16k-32b.pcm
```

- Soft-decode a AAC file to audio RAW PCM:

```
ffmpeg -i test1-16k-32b.aac -f s32le -acodec pcm_s32le -ar 16000 -ac 2 output-test1-16k-32b.pcm
```

2.6.13. Audio Soft Encoding

The topic provides examples for soft encoding an audio file.

The following code block shows the example in different scenarios.

- Soft-encode a RAW PCM file to WAV:

```
ffmpeg -f s32le -ac 2 -ar 16000 -i output-test1-16k-32b.pcm -acodec pcm_s32le output-test1-16k-32b.wav
```

- Soft-encode a RAW PCM file to AAC:

```
ffmpeg -f s32le -ac 2 -ar 16000 -i output-test1-16k-32b.pcm output-test1-16k-32b.aac
```

- Soft-encode a RAW PCM file to MP3:

```
ffmpeg -f s32le -ac 2 -ar 16000 -i output-test1-16k-32b.pcm output-test1-16k-32b.mp3
```

2.6.14. Audio Transcoding

The topic provides examples for transcoding an audio file.

The following code block shows the example in different scenarios.

- Transcode from WAV to AAC:

  ```
  ffmpeg -i test1-16k-32b.wav -acodec aac test1-16k-32b.aac
  ```

- Transcode from WAV to MP3:

  ```
  ffmpeg -i test1-16k-32b.wav -acodec libmp3lame test1-16k-32b.mp3
  ```

- Transcode from WAV to OGG Vorbis:

  ```
  ffmpeg -i test1-16k-32b.wav -acodec libvorbis test1-16k-32b.ogg
  ```

- Transcode from WAV to FLAC:

  ```
  ffmpeg -i test1-16k-32b.wav -acodec flac test1-16k-32b.flac
  ```

- Transcode from sample rate 16 K to 32 K:

  ```
  ffmpeg -i test1-16k-32b.mp3 -ar 32000 output-test1-32k-32b.mp3
  ```

- Transcode from 2-channel to 1-channel:

  ```
  ffmpeg -i test1-16k-32b.mp3 -ac 1 output-test1-16k-32b-ac1.mp3
  ```

- Transcode bit rate:

  ```
  ffmpeg -i test1-16k-32b.mp3 -ab 128k output-test1-16k-32b-ab128k.mp3
  ```

2.6.15. Content Cropping

The topic provides examples for cropping contents from an audio file or from a video file.

The following code block shows the example in different scenarios.

- Crop H.264 content from an H.264 file:

  ```
  ffmpeg -i 1080p.h264 -ss 00:00:10 -t 01:00:00 -vcodec copy -acodec copy 1080p.h264
  ```

- Crop content from an audio file beginning at time 00:30:00:

  ```
  ffmpeg -i test1-16k-32b.mp3 -ss 00:30:00 -t 00:01:00 -acodec copy output-test1-16k-32b-cut.mp3
  ```

2.6.16. Image Soft Encoding and Transcoding

The topic provides examples for soft encoding an image file and transcoding.

The following code block shows the example in different scenarios.

- Encode a YUV file to JPG:

  ```
  ffmpeg -s 1920*1080 -pix_fmt yuv444p -i output-big_big_image-1920-1080.yuv
   output-big_big_image-1920-1080.jpg
  ```

- Encode a YUV file to PNG:

  ```
  ffmpeg -s 1280*720 -pix_fmt yuv444p -i output-big_big_image-1280-720.yuv
   out-put-big_big_image-1280-720.png
  ```

- Encode a YUV file to BMP:

  ```
  ffmpeg -s 640*480 -pix_fmt yuv444p -i output-big_big_image-640-480.yuv output-big_big_image-640-480.bmp
  ```

- Encode a RGB file to JPG:

```
ffmpeg -s 640*960 -pix_fmt rgba -i output_640_960.rgb output_640_960.jpg
```

- Encode a RGB file to PNG:

```
ffmpeg -s 640*960 -pix_fmt rgba -i output_640_960.rgb output_640_960.png
```

- Encode a RGB file to BMP:

```
ffmpeg -s 640*960 -pix_fmt rgba -i output_640_960.rgb output_640_960.bmp
```

- Transcode a JPG file to PNG:

```
ffmpeg -i big_big_image.jpg big_big_image.png
```

- Transcode a JPG file to BMP:

```
ffmpeg -i big_big_image.jpg big_big_image.bmp
```

- Transcode and scale to a specific size:

```
ffmpeg -i big_big_image.jpg -vf scale=320:240 big_big_image_320_240.jpg
```

- Transcode and scale to a specific ratio:

```
ffmpeg -i big_big_image.jpg -vf scale=320:-1 big_big_image_zoom.jpg
```

- Decode a video file to image sequence:

```
ffmpeg -i 1080p_h264_aac_cut5s.mp4 -t 5 -s 1920*1080 output-cut-img-%3d.jpg
```

- Encode an image sequence to video:

```
ffmpeg -i output-cut-img-%3d.jpg output-jpg.mp4
```

2.6.17. Subtitling

The topic provides examples for adding subtitles to a video file.

The following code block shows the example in different scenarios.

- Add subtitles to an MP4 file:

```
ffmpeg -i 1080p_h264_aac.mp4 -i test_subtitle.srt -c copy -c:s mov_text out-put-sub-1080p_h264_aac.mp4
```

- Add subtitles to an MKV file:

```
ffmpeg -i 1080p_h264_aac.mkv -i test_subtitle.srt -c copy output-sub-1080p_h264_aac.mkv
```

2.6.18. Adding Logos

The topic provides an example for adding a company or product logo to a video file.

The following code block shows the command in detail.

```
ffmpeg -i 1080p_h264_aac_cut5s.mp4 -i logo_320_240.jpg -filter_complex overlay
 output-1080p_h264_aac_cut5s_logo.mp4
```

2.6.19. Playing Videos

The topic provides examples for playing a video file in the Debian operating system.

The following code block shows the example in different scenarios.

- Play an H.265 video file:

```
ffplay -v verbose -vcodec hevc_omx -i /test1_720P.h265
```

- Play an MP4 video file:

```
ffplay -v verbose -vcodec hevc_omx -f alsa hw:0,1 /720p_h265_aac.mp4
```

- Play an audio file:

```
ffplay -f s16le -ar 32000 -ac 2 test.pcm
```

2.7. Interface Description

FFmpeg supports the following libraries (application interfaces):

- **libavcodec**: encoding/decoding library
- **libavfilter**: graph-based frame editing library
- **libavformat**: I/O and muxing/demuxing library
- **libavdevice**: special devices muxing/demuxing library
- **libavutil**: common utility library
- **libswresample**: audio re-sampling, format conversion and mixing
- **libpostproc**: post processing library
- **libswscale**: color conversion and scaling library

The StarFive JH7110 SDK supports the standard FFmpeg API library, for more detailed information on each application interface, see FFmpeg Official Documentation.

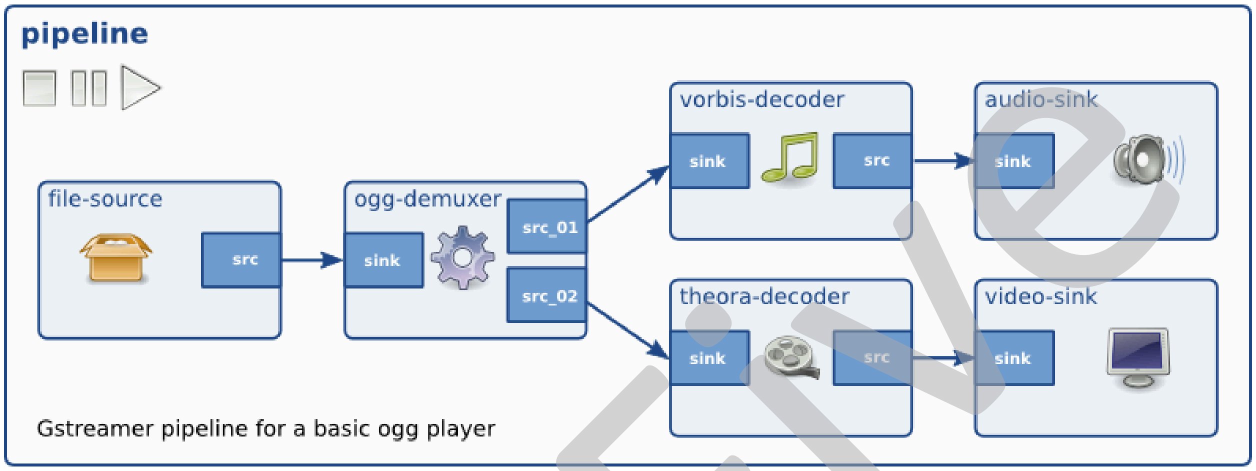
3. GStreamer

GStreamer is a pipeline-based multimedia framework that links together a wide variety of media processing systems to complete a complex workflow.

GStreamer supports a wide variety of media-handling components, including simple audio playback, audio and video playback, recording, streaming and editing.

The following figure shows the GStreamer framework.

Figure 3-1 GStreamer Framework



3.1. Source Code Location

Locate the JH7110 *Software Development Kit (SDK)* with the following information.

- **Repository**: https://github.com/starfive-tech/VisionFive2
- **Branch**: JH7110_VisionFive2_devel
- **Tag**: VF2_v2.4.4

The source code of GStreamer and its related plug-ins can be downloaded from the network, and then generated by adding the patches provided by StarFive. For details, please refer to https://github.com/starfive-tech/sft-riscv-buildroot/tree/JH7110_VisionFive2_devel/package/gstreamer1.

The source code of GStreamer and its required plug-ins are in the following directory: `buildroot/package/gstreamer1/`.

3.2. Plug-in Reference

GStreamer includes the following **gst-omx** plug-ins:

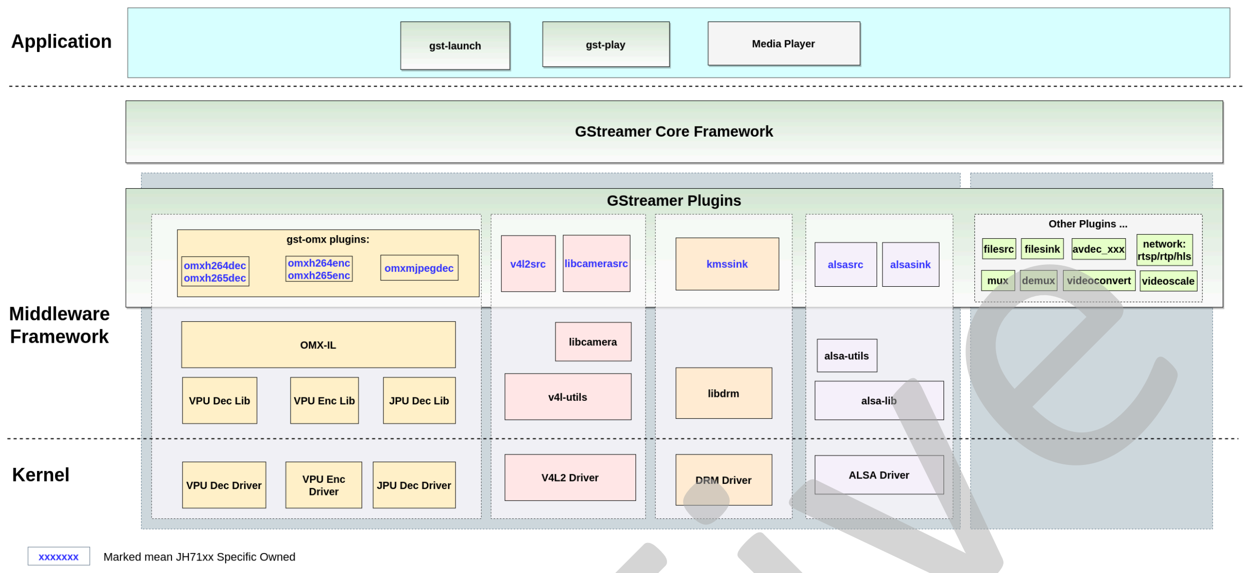
Table 3-1 GStreamer Plug-in Reference

Plug-in	Description
omxh265dec	OpenMAX IL H.265 Video Decoder
omxh265enc	OpenMAX IL H.265/AVC video Encoder
omxh264dec	OpenMAX IL H.264 Video Decoder
omxmjpegdec	OpenMAX IL MJPEG Video Decoder

3.3. Block Diagram

The following figure shows the block diagram of GStreamer.

Figure 3-2 GStreamer Block Diagram



3.4. Build GStreamer for Buildroot

This section introduces the procedure of building GStreamer for buildroot.

1. Enable the related macros (which are enabled by default).

```
BR2_PACKAGE_MPP=y
BR2_PACKAGE_MPP_ALLOCATOR_DRM=y
BR2_PACKAGE_GSTREAMER1_ROCKCHIP=y
BR2_PACKAGE_LINUX_RGA=y
BR2_PACKAGE_CA_CERTIFICATES=y
BR2_PACKAGE_LIBSOUP_SSL=y
BR2_PACKAGE_GSTREAMER1=y
BR2_PACKAGE_GST1_PLUGINS_BASE=y
BR2_PACKAGE_GST1_PLUGINS_BASE_PLUGIN_ALSA=y
BR2_PACKAGE_GST1_PLUGINS_BASE_PLUGIN_VIDEOCONVERT=y
BR2_PACKAGE_GST1_PLUGINS_BASE_PLUGIN_VIDEOTESTSRC=y
BR2_PACKAGE_GST1_PLUGINS_GOOD=y
BR2_PACKAGE_GST1_PLUGINS_GOOD_PLUGIN_AUDIOPARSERS=y
BR2_PACKAGE_GST1_PLUGINS_GOOD_PLUGIN_AUTODETECT=y
BR2_PACKAGE_GST1_PLUGINS_GOOD_PLUGIN_DEINTERLACE=y
BR2_PACKAGE_GST1_PLUGINS_GOOD_PLUGIN_FLV=y
BR2_PACKAGE_GST1_PLUGINS_GOOD_PLUGIN_GDKPIXBUF=y
BR2_PACKAGE_GST1_PLUGINS_GOOD_PLUGIN_MATROSKA=y
BR2_PACKAGE_GST1_PLUGINS_GOOD_PLUGIN_MPG123=y
BR2_PACKAGE_GST1_PLUGINS_GOOD_PLUGIN_SOUPHTTPSRC=y
BR2_PACKAGE_GST1_PLUGINS_BAD=y
BR2_PACKAGE_GST1_PLUGINS_BAD_PLUGIN_DVBSUBOVERLAY=y
BR2_PACKAGE_GST1_PLUGINS_BAD_PLUGIN_DVDSPU=y
BR2_PACKAGE_GST1_PLUGINS_BAD_PLUGIN_JPEGFORMAT=y
BR2_PACKAGE_GST1_PLUGINS_BAD_PLUGIN_KMS=y
BR2_PACKAGE_GST1_PLUGINS_BAD_PLUGIN_MPEGDEMUX=y
BR2_PACKAGE_GST1_PLUGINS_BAD_PLUGIN_MPEG2ENC=y
BR2_PACKAGE_GST1_PLUGINS_BAD_PLUGIN_VIDEOPARSERS=y
BR2_PACKAGE_GST1_PLUGINS_BAD_PLUGIN_ADPCMDEC=y
BR2_PACKAGE_GST1_PLUGINS_BAD_PLUGIN_ADPCMENC=y
BR2_PACKAGE_GST1_PLUGINS_BAD_PLUGIN_FAAD=y
BR2_PACKAGE_GST1_PLUGINS_UGLY=y
BR2_PACKAGE_GST1_PLUGINS_UGLY_PLUGIN_ASFDEMUX=y
BR2_PACKAGE_GST1_PLUGINS_UGLY_PLUGIN_DVDLPCMDEC=y
```

```
BR2_PACKAGE_GST1_PLUGINS_UGLY_PLUGIN_DVDSUB=y
BR2_PACKAGE_GST1_PLUGINS_UGLY_PLUGIN_MPEG2DEC=y
...
```

The complete list of plug-ins can be found in **menuconfig > Target packages > Audio and video applications > gstreamer 1.x.** .

2. Build them in the SDK root directory directly with the following command.

```
$ make
```

3.5. Build GStreamer for Debian

The source code should be placed on the board, and make sure that the `debian/` directory exists in the root directory of the source code.

Enter the source root directory and execute the following procedure:

1. Use the following command to update software resources.

```
apt update
```

2. Use the following command to install library dependencies.

```
apt build-dep
```

3. (Optional) Use the following command to start build the Debian installation package.

```
dpkg-buildpackage -b -d -uc -us
```

Result: After the building is completed, the Debian installation package will be generated in the upper directory, which can be installed by using `dpkg -i xxx.deb`.

4. Build and install.

```
meson build && ninja -C build install
```

It is generally recommended to use the first way to build the Debian installation package, which can ensure that the options such as compilation and installation are unified.

Note:
Some compilation options depend on the macro definitions in header files such as `video-format.h`, so you need to install the `libgstreamer-plugins-base1.0-dev` package first to ensure the headers such as `video-format.h` to the latest and ensure that certain features are turned on.

3.6. General Commands

The chapter describes the command tools used in general for GStreamer.

Before you start using the commands, make sure you have run the following command to check the version of the GStreamer.

```
gst-inspect-1.0 --version
```

3.6.1. gst-launch-1.0

The command `gst-launch-1.0` works as the GStreamer launcher for building pipelines quickly.

The following code block shows an example of generating a video by **videotestsrc**, and play it through **xvimagesink**.

```
gst-launch-1.0 videotestsrc ! xvimagesink
```

The command is part of the standard GStreamer command line tools, see GStreamer Official Documentation for more information on how to use the command.

3.6.2. GS Version Check

Use the following command to check your current GStreamer version.

```
gst-inspect-1.0 --version
```

3.6.3. gst-play-1.0

The command `gst-play-1.0` works as the GStreamer player for playing various streaming media files.

The following code block shows an example of playing a test video file through **xvimagesink**.

```
gst-play-1.0 test.mp4 --videosink=xvimagesink
```

The command is part of the standard GStreamer command line tools, see GStreamer Official Documentation for more information on how to use the command.

3.6.4. gst-inspect-1.0

The command `gst-inspect-1.0` works as the GStreamer inspector for listing all the plug-in information in detail.

The following code block shows an example of listing all plug-ins details.

```
gst-inspect-1.0
```

The following code block shows an example of listing all details of the **xvimagesink** plug-in.

```
gst-inspect-1.0 xvimagesink
```

The command is part of the standard GStreamer command line tools, see GStreamer Official Documentation for more information on how to use the command.

3.6.5. Enable Logs

To enable the log function for GStreamer, follow the procedure below.

1. Set environment variables.

   ```
   export GST_DEBUG=2
   ```

   Or specified before the command, and invalid after the end of the command.

   ```
   GST_DEBUG=2 gst-play-1.0
   ```

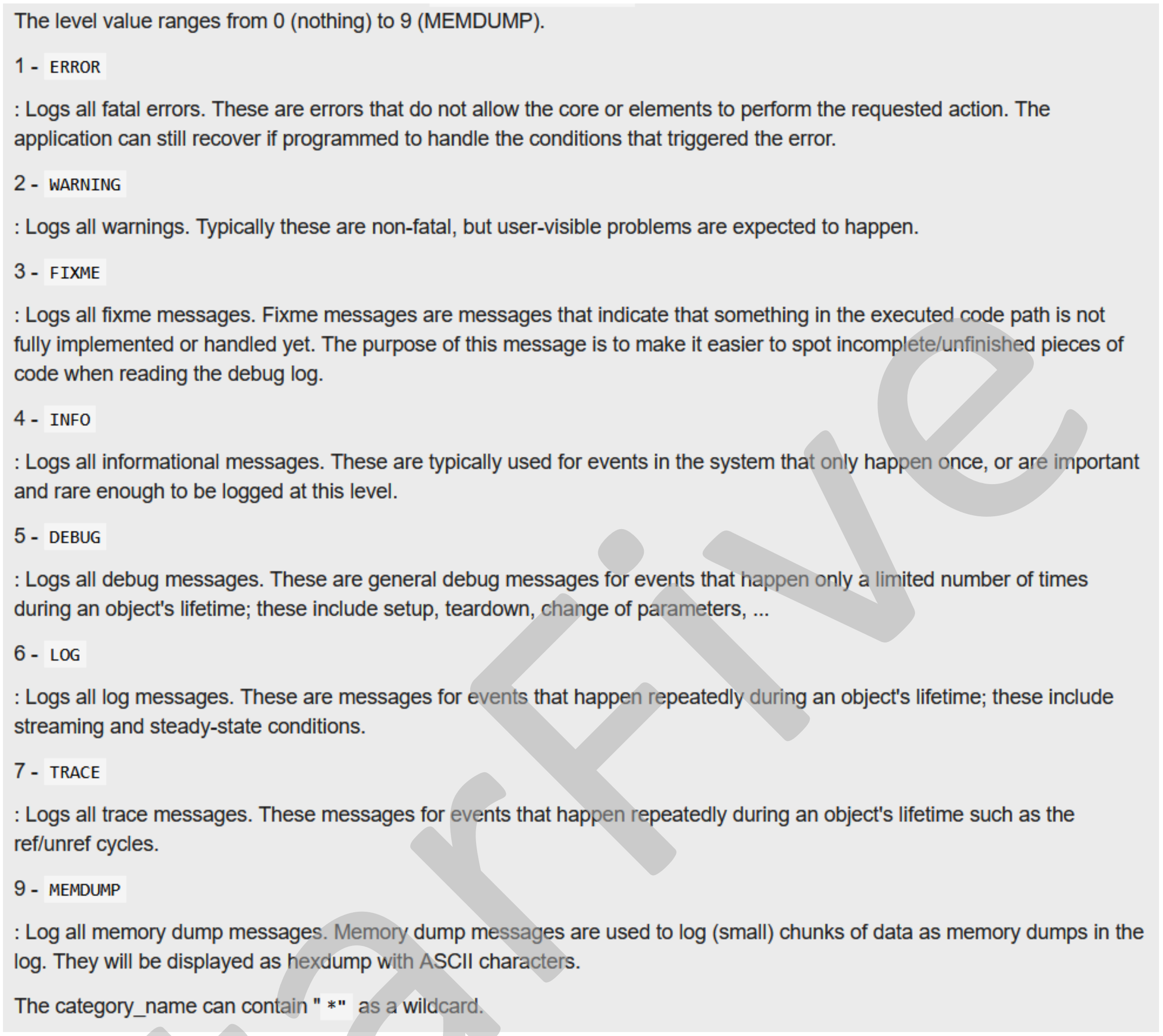
2. Specify the log levels for each module.

   The following code block shows an example, **fpsdisplaysink** is specified as DEBUG (5), **xvimage*** is specified as FIXME (3), and the others are specified as WARNING (2).

   ```
   GST_DEBUG=2,fpsdisplaysink:5,xvimage*:3
   ```

The following screen shows the GStreamer log levels in detail. For more information, see GStreamer Official Documentation on Environment Variables.

Figure 3-3 GStreamer Log Levels



The level value ranges from 0 (nothing) to 9 (MEMDUMP).

1 - `ERROR`

: Logs all fatal errors. These are errors that do not allow the core or elements to perform the requested action. The application can still recover if programmed to handle the conditions that triggered the error.

2 - `WARNING`

: Logs all warnings. Typically these are non-fatal, but user-visible problems are expected to happen.

3 - `FIXME`

: Logs all fixme messages. Fixme messages are messages that indicate that something in the executed code path is not fully implemented or handled yet. The purpose of this message is to make it easier to spot incomplete/unfinished pieces of code when reading the debug log.

4 - `INFO`

: Logs all informational messages. These are typically used for events in the system that only happen once, or are important and rare enough to be logged at this level.

5 - `DEBUG`

: Logs all debug messages. These are general debug messages for events that happen only a limited number of times during an object's lifetime; these include setup, teardown, change of parameters, ...

6 - `LOG`

: Logs all log messages. These are messages for events that happen repeatedly during an object's lifetime; these include streaming and steady-state conditions.

7 - `TRACE`

: Logs all trace messages. These messages for events that happen repeatedly during an object's lifetime such as the ref/unref cycles.

9 - `MEMDUMP`

: Log all memory dump messages. Memory dump messages are used to log (small) chunks of data as memory dumps in the log. They will be displayed as hexdump with ASCII characters.

The category_name can contain " `*`" as a wildcard.

3.7. Common Plug-ins

3.7.1. Source

The source plug-ins are used to generate data.

3.7.1.1. filesrc

The plug-in **filesrc** is used to read data from a file.

The following code block provides an example.

```
gst-launch-1.0 filesrc location=/tmp/test ! filesink location=/tmp/test2
```

The plug-in is part of the standard GStreamer plug-in tools, see GStreamer Official Documentation on Plug-in for more information on how to use the plug-in.

3.7.1.2. videotestsrc

The plug-in **videotestsrc** is used to generate video data.

The following code block provides an example of generating video output in the default format.

```
gst-launch-1.0 videotestsrc ! xvimagesink
```

The following code block provides an example of generating video output in a specified format (NV12) with specified width and height.

```
gst-launch-1.0 videotestsrc ! "video/x-raw,width=1920,height=1080,format=(string)NV12" !xvimagesink
```

The plug-in is part of the standard GStreamer plug-in tools, see GStreamer Official Documentation on Plug-in for more information on how to use the plug-in.

3.7.2. Sink

The sink plug-ins are used to receive data.

3.7.2.1. filesink

The plug-in **filesink** is used to save the received data as a file.

The following code block provides an example.

```
gst-launch-1.0 filesrc location=/tmp/test ! filesink location=/tmp/test2
```

The plug-in is part of the standard GStreamer plug-in tools, see GStreamer Official Documentation on Plug-in for more information on how to use the plug-in.

3.7.2.2. fakesink

The plug-in **fakesink** is used to discard all the received data.

The following code block provides an example.

```
gst-launch-1.0 filesrc location=/tmp/test ! fakesink
```

The plug-in is part of the standard GStreamer plug-in tools, see GStreamer Official Documentation on Plug-in for more information on how to use the plug-in.

3.7.2.3. xvimagesink

The plug-in **xvimagesink** is used as the video sink to receive video and display, which is implemented by the X11 interface.

The following code block provides an example.

```
gst-launch-1.0 videotestsrc ! xvimagesink
```

The plug-in is part of the standard GStreamer plug-in tools, see GStreamer Official Documentation on Plug-in for more information on how to use the plug-in.

3.7.2.4. kmssink

The plug-in **kmssink** is used as the video sink to receive video and display. The plug-in is implemented through the KMS interface and requires an exclusive hardware decoding layer.

The following code block provides an example.

```
gst-launch-1.0 videotestsrc ! kmssink
```

The plug-in is part of the standard GStreamer plug-in tools, see GStreamer Official Documentation on Plug-in for more information on how to use the plug-in.

3.7.2.5. waylandsink

The plug-in **waylandsink** is used as the video sink to receive video and display. The plug-in is implemented through the wayland interface and requires an exclusive hardware decoding layer.

The following code block provides an example.

```
gst-launch-1.0 videotestsrc ! waylandsink
```

The plug-in is part of the standard GStreamer plug-in tools, see GStreamer Official Documentation on Plug-in for more information on how to use the plug-in.

3.7.2.6. rkximagesink

The plug-in **waylandsink** is used as the video sink to receive video and display. The plug-in is implemented through the wayland interface and requires an exclusive hardware decoding layer.

The following code block provides an example.

```
gst-launch-1.0 videotestsrc ! waylandsink
```

The plug-in is part of the standard GStreamer plug-in tools, see GStreamer Official Documentation on Plug-in for more information on how to use the plug-in.

3.7.2.7. fpsdisplaysink

The plug-in **waylandsink** is used as the video sink to receive video and display. The plug-in is implemented through the wayland interface and requires an exclusive hardware decoding layer.

The following code block provides an example.

```
gst-launch-1.0 videotestsrc ! waylandsink
```

The plug-in is part of the standard GStreamer plug-in tools, see GStreamer Official Documentation on Plug-in for more information on how to use the plug-in.

3.8. Use Examples

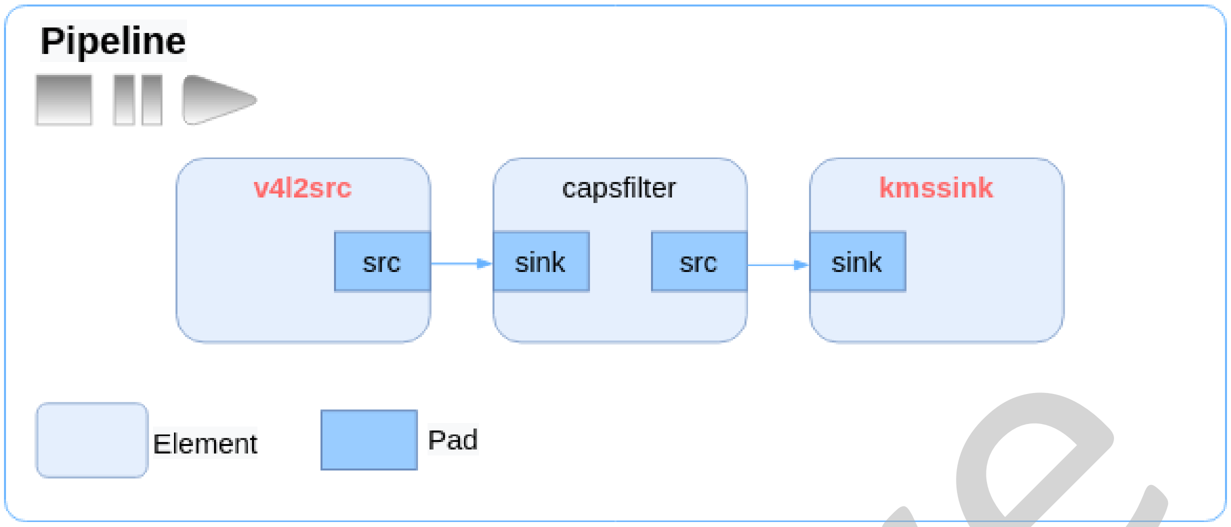
3.8.1. Video Capture

The following code block provides an example of capturing video content from camera/sensor and display over HDMI.

```
gst-launch-1.0 v4l2src device=/dev/video1 ! video/x-raw,format=NV12,width=1920,height=1080 ! kmssink
 driver-name=starfive force-modesetting=1
```

The following diagram shows the video capturing workflow.

Figure 3-4 GStreamer Video Capturing Workflow



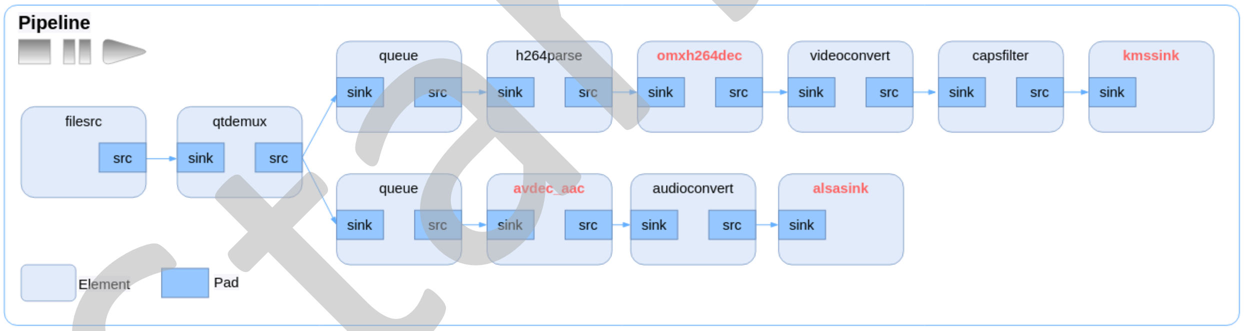
3.8.2. MP4 Playback

The following code block provides an example of playing back an MP4 video file, in this case, the H.264 video stream is decoded via the **gst-omx** plug-in.

```
gst-launch-1.0 -v filesrc location=1080p_h264_aac.mp4  ! qtdemux name=demux demux.audio_0 ! queue !
 avdec_aac ! audioconvert ! alsasink device=hw:0,2 demux.video_0 ! queue ! h264parse ! omxh264dec  ! video
 convert ! video/x-raw,width=1920,height=1080 ! kmssink driver-name=starfive force-modesetting=1
```

The following diagram shows the MP4 playback workflow.

Figure 3-5 GStreamer MP4 Playback Workflow



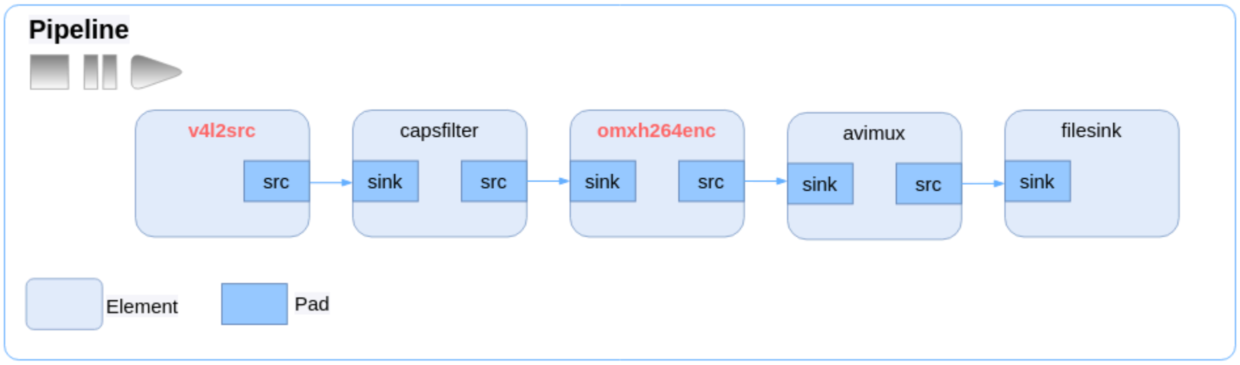
3.8.3. Video Recording

The following code block provides an example of encoding video stream content from camera/sensor into H.264 and encapsulating it to AVI format.

```
gst-launch-1.0 v4l2src device=/dev/video1 ! video/x-raw,width=640,height=480,format=NV12 !  omxh264enc !
 avimux ! filesink location=11.avi
```

The following diagram shows the video recording workflow.

Figure 3-6 GStreamer Video Recording Workflow



3.8.4. Specify Display in USDK

Please check the VOUT module related documents to see how the plane and connector ID is allocated.

The following list contains the examples for specifying the screen display in the *User Software Development Kit (USDK)*.

- The following code block provides an example of specifying display on a MIPI device.

```
gst-launch-1.0 -v filesrc location=4K_30FPS_AVC_MainL5_2.h265 ! h265parse ! omxh265dec ! videoconvert !
 videoscale ! video/x-raw, width=800, height=480,format=NV12 ! kmssink driver-name=starfive
 force-modesetting=1 plane-id=74 connector-id=120
```

- The following code block provides an example of specifying display on an HDMI device.

```
gst-launch-1.0 filesrc location=2k.18fps.mp4 ! qtdemux name=demux demux.video_0 ! queue !
 h265parse ! omxh265dec ! videoconvert ! videoscale ! video/x-raw,width=1920,height=1080 !  kmssink
 driver-name=starfive force-modesetting=1 plane-id=39 connector-id=116
```

- The following code block provides an example of capturing video contents and specifying display on an MIPI device.

```
gst-launch-1.0 v4l2src device=/dev/video1 ! video/x-raw,width=1920,height=1080 ! deinterlace !
 videoconvert ! videoscale !video/x-raw,width=800,height=480 !  kmssink driver-name=starfive
 force-modesetting=1 plane-id=74 connector-id=118
```

3.8.5. Specify Display Debian

The following code block provides an example of specifying display in the Debian *Operating System (OS)*.

```
gst-launch-1.0 filesrc location=file_example_MP4_640_3MG.mp4 ! qtdemux name=demux demux.video_0 ! queue !
 h264parse ! omxh264dec ! xvimagesink display=:0.0
```

3.8.6. Create Pipelines Automatically

By using the plug-in **decodebin** or **uridecodebin**, GStreamer can create correct pipelines automatically. The following code blocks provide some examples.

- The following code block provides an example of using the 3 plug-ins **filesrc**, **decodebin**, and **kmssink** together.

```
gst-launch-1.0 filesrc location=4K_h265_hevc_30FPS.mkv ! decodebin ! kmssink driver-name=starfive
 force-modesetting=1
```

- The following code block provides an example of using the 2 plug-ins **uridecodebin** and **kmssink** together. The location of the media file is recorded as value of the URI property of **uridecodebin**.

```
gst-launch-1.0 uridecodebin uri=file:///media/root/4K_h265_hevc_30FPS.mkv ! kmssink
 driver-name=starfive force-modesetting=1
```

- The following code block provides an example of using the 2 plug-ins **uridecodebin** and **kmssink** together. The location of the media file is recorded as value of the URI property of **uridecodebin** and output is forced to the MIPI device.

```
gst-launch-1.0 uridecodebin uri=file:///media/root/640_360_h265_aac.mkv ! kmssink driver-name=starfive
 force-modesetting=1 plane-id=74 connector-id=118
```

- The following code block provides an example of using the plug-in **uridecodebin** to decode a media file, but only play audio from the decoded file.

```
gst-launch-1.0 uridecodebin uri=file:///media/root/640_360_h265_aac.mkv ! audioconvert ! alsasink
 device=hw:0,1
```

3.8.7. Play Camera Recordings in Debian

Follow the steps below to enable the screen to play real-time camera recordings in the Debian *Operating System (OS)*.

1. Install the **v4l2-ctl** tool to detect the recordings and list the resolutions supported by the current camera.
2. Select one of the resolutions for display.

   The following code blocks provide some examples.

   - Set the resolution to 800 × 600:

   ```
   gst-launch-1.0 v4l2src device=/dev/video4 ! video/x-raw,width=800,height=600 ! videoconvert !
    xvimagesink
   ```

   - Set the resolution to 1280 × 720:

   ```
   gst-launch-1.0 v4l2src device=/dev/video4 ! video/x-raw,width=1280,height=720 ! videoconvert !
    xvimagesink
   ```

   - Set the resolution to 1280 × 900:

   ```
   gst-launch-1.0 v4l2src device=/dev/video4 ! video/x-raw,width=1280,height=960 ! videoconvert !
    xvimagesink
   ```

3.8.8. Disable Audio Output

The following code block provides an example of disabling audio output.

```
gst-play-1.0 -v 1080p_h264_aac.mp4 --audiosink=fakesink --videosink='kmssink driver-name=starfive
 force-modesetting=1'
```

3.8.9. Specify Audio Output Device

The following list contains example code blocks for specifying the audio output device.

-
  ```
  gst-play-1.0 1080p_h264_aac.mp4 --audiosink='alsasink device=hw:0,1' --videosink='kmssink
   driver-name=starfive force-modesetting=1'
  ```

-
  ```
  gst-play-1.0 1080p_h264_aac.mp4 --audiosink='alsasink device=hw:0,2' --videosink='kmssink
   driver-name=starfive force-modesetting=1'
  ```

3.9. FAQ

3.9.1. Resources Cannot Be Played

Problem

Some sources cannot be played, and the system LOG is lagging and the progress is not printed or the progress is always 0.

Solution

Try to use the playbin3 plug-in tool. The following code block provides an example:

```
gst-play-1.0 --flags=3 --use-playbin3 test.mp4
```

3.9.2. Played with no Sound

Problem

Some sources can be played, with only display but no sound output.

Solution

You can try to specify the **audiosink** manually. The following code block provides an example.

```
 gst-play-1.0 --flags=3 test.mp4 --audiosink="alsasink device=hw:0,0"
```

It is recommended to make sure the video can be played properly using basic testing tools such as **aplay** and then use GStreamer to debug.

3.9.3. Play Failed via SSH

Problem

In the Debian Operating System (OS), playing some audio or video sources may fail via SSH but succeed on the UART terminal.

Solution

After log in through the SSH terminal each time, you can try to check if the DISPLAY environment variable are properly configured to specify the display device.

The following code block provides an example.

```
export DISPLAY=:0
gst-play-1.0 1080p_h264_aac.mp4
```

3.9.4. Played Slowly

Problem

The audio or the video files are played slowly or even suspended.

Solution

It may become when display or audio play is slower than video/audio decoder. Be always use queue plugin as buffer before **alsasink** and **kmssink**.

- For H.264, try the following:
    - The following code block provides an example of setting the audio format manually.

      ```
      gst-launch-1.0 -v filesrc location=640_360_h264_aac.mkv ! matroskademux name=demux
       demux.audio_0 ! aacparse ! avdec_aac ! audioconvert ! audioresample ! queue ! alsasink
       device=hw:0,1 demux.video_0 ! queue ! h264parse ! omxh264dec ! videoconvert ! videoscale !
       video/x-raw,width=800,height=480,format=NV12 ! queue ! kmssink driver-name=starfive
       force-modesetting=1 plane-id=74 connector-id=118
      ```

    - The following code block provides an example of using the **decodebin** plug-in to identify audio format automatically.

      ```
      gst-launch-1.0 -v filesrc location=640_360_h264_aac.mkv ! matroskademux name=demux
       demux.audio_0 ! decodebin ! audioconvert ! audioresample ! queue ! alsasink device=hw:0,1
      ```

```
demux.video_0 ! queue ! h264parse ! omxh264dec ! videoconvert ! videoscale !
video/x-raw,width=800,height=480,format=NV12 ! queue ! kmssink driver-name=starfive
force-modesetting=1 plane-id=74 connector-id=118
```

- For H.265, try the following:
    - The following code block provides an example of setting the audio format manually.

    ```
    gst-launch-1.0 -v filesrc location=640_360_h265_aac.mkv ! matroskademux name=demux
     demux.audio_0 ! aacparse ! avdec_aac ! audioconvert ! audioresample ! queue ! alsasink
     device=hw:0,1 demux.video_0 ! queue ! h265parse ! omxh265dec ! videoconvert ! videoscale !
     video/x-raw,width=800,height=480,format=I420 ! queue ! kmssink driver-name=starfive
     force-modesetting=1 plane-id=74 connector-id=118
    ```

    - The following code block provides an example of using the **decodebin** plug-in to identify audio format automatically.

    ```
    gst-launch-1.0 -v filesrc location=640_360_h265_aac.mkv ! matroskademux name=demux
     demux.audio_0 ! decodebin ! audioconvert ! audioresample ! queue ! alsasink device=hw:0,1
     demux.video_0 ! queue ! h265parse ! omxh265dec ! videoconvert ! videoscale !
     video/x-raw,width=800,height=480,format=NV12 ! queue ! kmssink driver-name=starfive
     force-modesetting=1 plane-id=74 connector-id=118
    ```

3.9.5. Copy MP4 Invalid

Problem

The system generates invalid mp4 file while end with “CTRL + C” .

Solution

According to MOV/MP4 file format, it would lack of moov meta data while end with “CTRL + C”. You should add the **num-buffers** property when the MOV/MP4 file is created by **audiotestsrc** or **videotestsrc**.

The following code block provides an example.

```
gst-launch-1.0 -v videotestsrc ! video/x-raw,format=NV12,width=640,height=480 ! filesink location=480P.yuv
```

```
gst-launch-1.0 filesrc location=480P.yuv ! rawvideoparse width=640 height=480 format=nv12 ! queue !
 omxh265enc ! h265parse ! mux. audiotestsrc num-buffers=440 ! audioconvert ! avenc_aac ! queue ! mux. qtmux
 name=mux ! filesink location=test.mp4
```

3.9.6. Play Failure on the Start

Problem

Play is failed and the player freezes at the first frame of the video.

Solution

Use the following procedure to debug.

1. Make sure whether the correct plug-in is used. It may occur while the pipeline is created successfully, but using an incorrect plug-in. The following code blocks provide some examples of detecting its audio/video format.

    -
      ```
      gst-discoverer-1.0 4K_h265_hevc_30FPS.mkv
      ```

    -
      ```
      mediainfo 4K_h265_hevc_30FPS.mp4
      ```

    -
      ```
      ffmpeg -i 4K_h265_hevc_30FPS.mkv
      ```
2. Try to play with the **decodebin** or the **uridecodebin** plug-in. They can always find the correct plug-ins and create pipelines automatically.

   The following list shows some example code blocks.

-
  ```
  gst-launch-1.0 filesrc location=4K_h265_hevc_30FPS.mkv ! decodebin ! kmssink
   driver-name=starfive force-modesetting=1
  ```

-
  ```
  gst-launch-1.0 uridecodebin uri=file:///media/root/4K_h265_hevc_30FPS.mkv ! kmssink
   driver-name=starfive force-modesetting=1
  ```

4. OpenMAX

OpenMAX is a royalty-free, cross-platform API that provides comprehensive streaming media codec and application portability by enabling accelerated multimedia components to be developed, integrated and programmed across multiple operating systems and silicon platforms.

4.1. Source Code Location

Locate the JH7110 *Software Development Kit (SDK)* with the following information.

- **Repository**: https://github.com/starfive-tech/VisionFive2
- **Branch**: JH7110_VisionFive2_devel
- **Tag**: VF2_v2.4.4

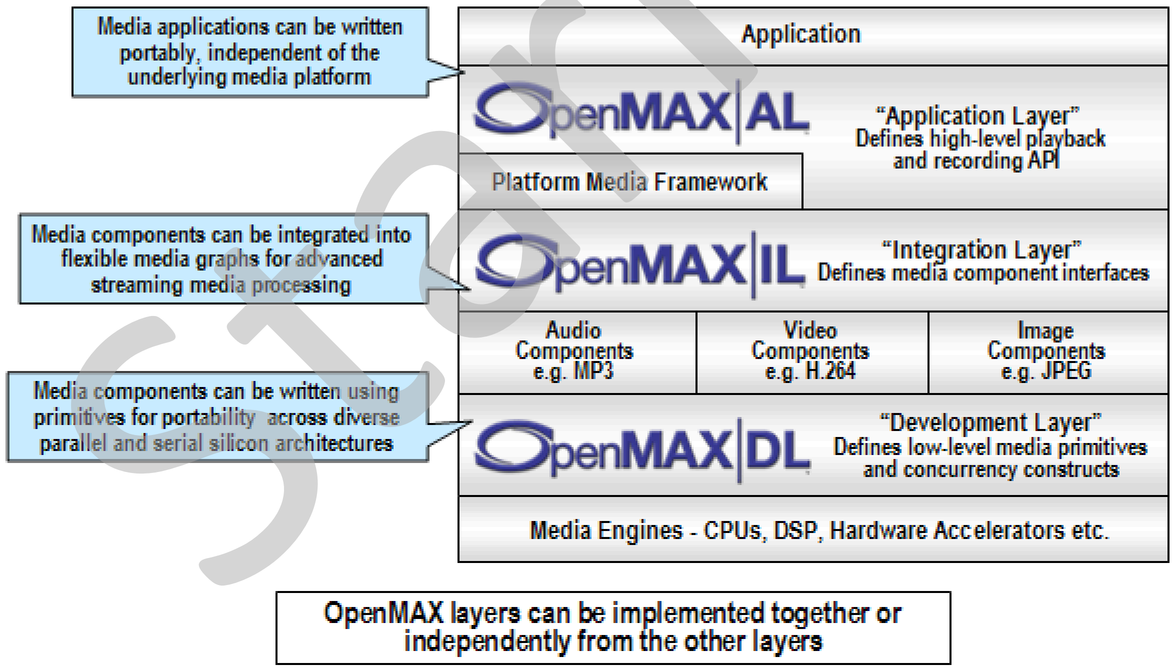
The source code of OpenMAX and its related plug-ins can be downloaded from the network, and then generated by adding the patches provided by StarFive. For details, please refer to https://github.com/starfive-tech/sft-riscv-buildroot/tree/JH7110_VisionFive2_devel/package/openmax.

The source code of OpenMAX and its required plug-ins are in the following directory: `buildroot/package/openmax/`.

4.2. Block Diagram

The following figure shows the block diagram of the OpenMAX module.

Figure 4-1 OpenMax Block Diagram


Media applications can be written portably, independent of the underlying media platform
Media components can be integrated into flexible media graphs for advanced streaming media processing
Media components can be written using primitives for portability across diverse parallel and serial silicon architectures
Application
OpenMAX AL
"Application Layer" Defines high-level playback and recording API
Platform Media Framework
OpenMAX IL
"Integration Layer" Defines media component interfaces
Audio Components e.g. MP3
Video Components e.g. H.264
Image Components e.g. JPEG
OpenMAX DL
"Development Layer" Defines low-level media primitives and concurrency constructs
Media Engines - CPUs, DSP, Hardware Accelerators etc.
OpenMAX layers can be implemented together or independently from the other layers

JH7110 SDK use OpenMAX IL layer only.

- **OpenMAX IL**: Integration Layer

OpenMAX IL serves as a low-level interface for audio, video, and imaging codecs used in embedded and/or mobile devices. It gives applications and media frameworks the ability to interface with multimedia codecs and supporting components (i.e., sources and sinks) in a unified manner. The codecs themselves may be any combination of hardware or software and are completely transparent to the user. Without a standardized interface of this nature, codec vendors must write to proprietary or closed interfaces to integrate into mobile devices. The principal goal of the IL is to give codecs a degree of system abstraction using a specialized arsenal of features, honed to combat the problem of portability among many vastly different media systems.

4.3. Use Examples

The OpenMAX *Integration Layer (IL)* API allows integration layer clients to control multimedia components in the audio, video and image domains. An "other" domain is also included to provide for extra functionality, such as audio-video (A/V) synchronization. The user of the OpenMAX Integration Layer API is usually a multimedia framework. In the rest of this document, the user of the OpenMAX Integration Layer API will be referred to as the IL client. The OpenMAX Integration Layer API is defined in a set of header files, namely:

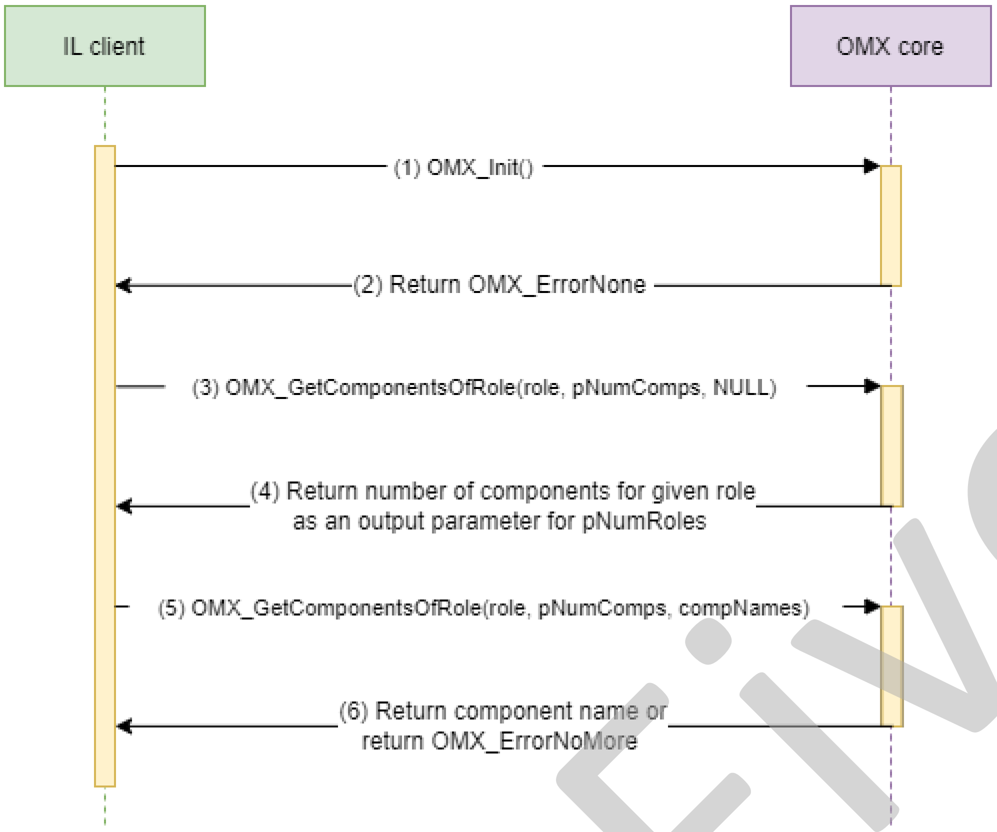
- `OMX_Types.h`: Data types used in the OpenMAX IL
- `OMX_Core.h`: OpenMAX IL core API
- `OMX_Component.h`: OpenMAX IL component API
- `OMX_Audio.h`: OpenMAX IL audio domain data structures
- `OMX_IVCommon.h`: OpenMAX IL structures common to image and video domains
- `OMX_Video.h`: OpenMAX IL video domain data structures
- `OMX_Image.h`: OpenMAX IL image domain data structures
- `OMX_Other.h`: OpenMAX IL other domain data structures (includes A/V synchronization)
- `OMX_Index.h`: Index of all OpenMAX IL-defined data structures
- `OMX_ContentPipe.h`: Content pipe definition

4.3.1. Determine Role Support

To determine whether the OMX core supports the required role, the client can enumerate the component name based on the role from the OpenMAX core. If no component supports the given role, the output parameter for the **OMX_GetRolesOfComponent** is 0.

The following diagram shows the role support determination workflow.

Figure 4-2 Role Support Determination Workflow


IL client
OMX core
(1) OMX_Init()
(2) Return OMX_ErrorNone
(3) OMX_GetComponentsOfRole(role, pNumComps, NULL)
(4) Return number of components for given role as an output parameter for pNumRoles
(5) OMX_GetComponentsOfRole(role, pNumComps, compNames)
(6) Return component name or return OMX_ErrorNoMore

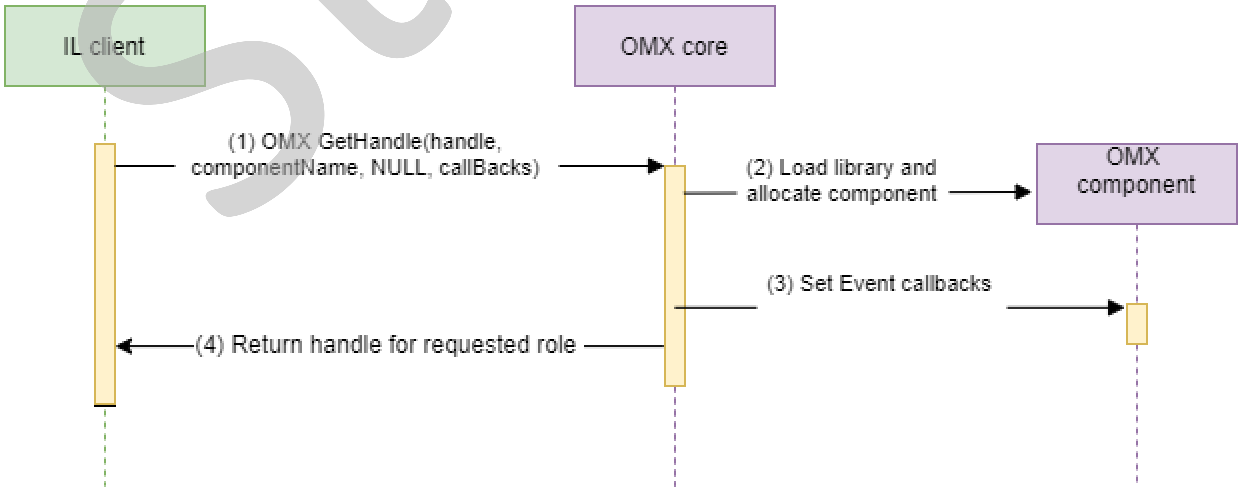
4.3.2. Load Component

The following items correspond to the steps in Figure :

1. The **OMX_GetHandle** call loads the component library dynamically and initializes the component as (2) in the following diagram.
2. Callback events are set to directly notify the client for **FillBufferDone**, **EmptyBufferDone**, and events from the component as (3) in the following diagram.

The following diagram shows the component loading workflow.

Figure 4-3 Component Loading Workflow


IL client
OMX core
OMX component
(1) OMX_GetHandle(handle, componentName, NULL, callBacks)
(2) Load library and allocate component
(3) Set Event callbacks
(4) Return handle for requested role

4.3.3. Component Handshake

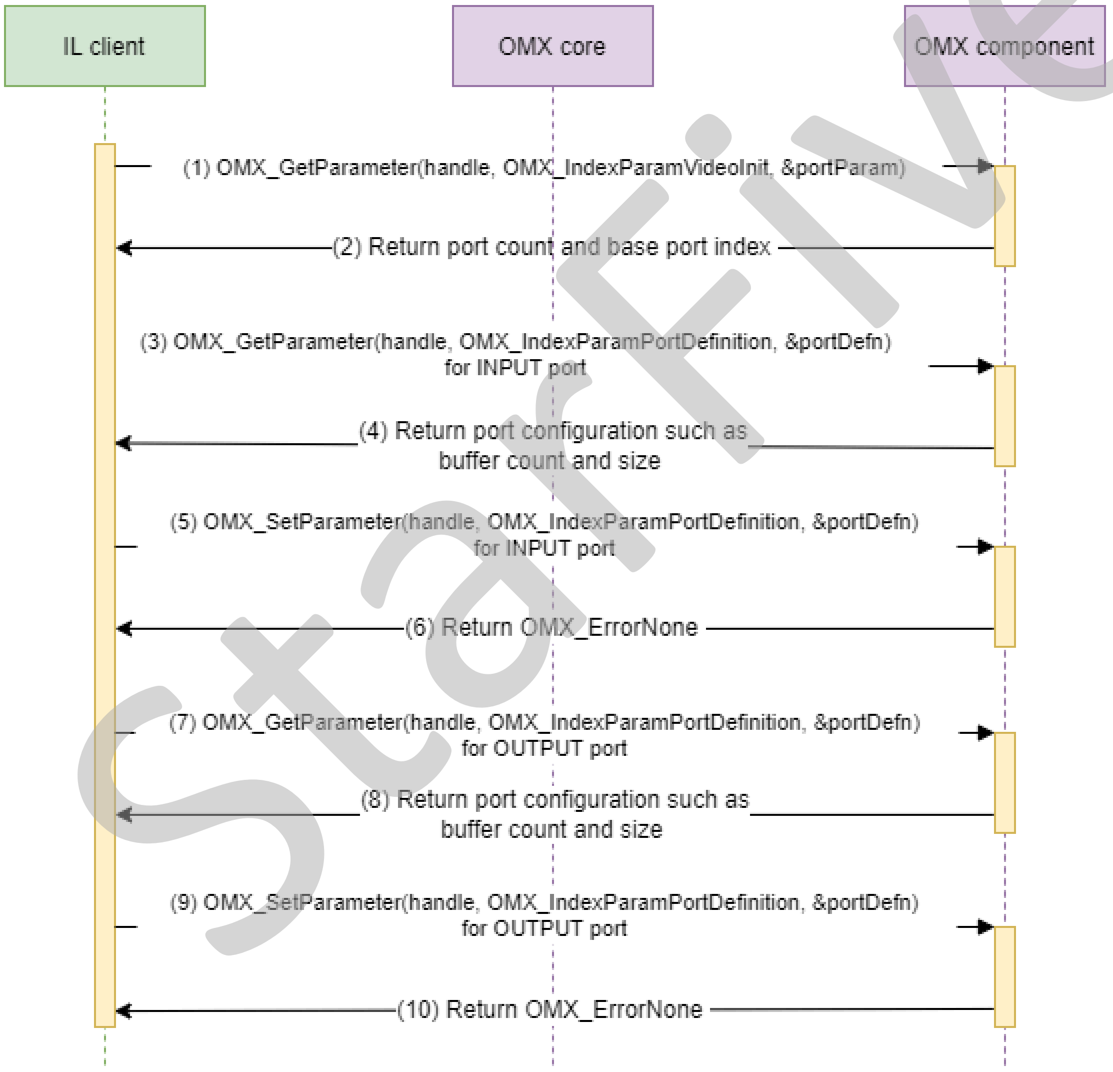
After the component handle is acquired, the next step is to initialize and configure the component.

To initialize and configure the component, the client can retrieve the port definition (**OMX_PARAM_PORTDEFINITIONTYPE**) and format of the components and set them accordingly.

1. **OMX_GetParameter** (with **OMX_IndexVideoInit**) is called (1) to retrieve the number of ports supported by the OMX component
2. The component returns the number of ports supported as well as the base port index to retrieve the individual port configuration as (2) in the following diagram.
3. For each port, the client can get information about the preferred buffer size, number of buffers, etc., of the component (3) and (7) in the following diagram.
4. The IL client can call **OMX_SetParameter** to configure the port with either the configuration retrieved as (4) and (8) in the following diagram, or with a different configuration.

The following diagram shows the component handshake workflow.

Figure 4-4 Component Handshake Workflow


IL client
OMX core
OMX component
(1) OMX_GetParameter(handle, OMX_IndexParamVideoInit, &portParam)
(2) Return port count and base port index
(3) OMX_GetParameter(handle, OMX_IndexParamPortDefinition, &portDefn) for INPUT port
(4) Return port configuration such as buffer count and size
(5) OMX_SetParameter(handle, OMX_IndexParamPortDefinition, &portDefn) for INPUT port
(6) Return OMX_ErrorNone
(7) OMX_GetParameter(handle, OMX_IndexParamPortDefinition, &portDefn) for OUTPUT port
(8) Return port configuration such as buffer count and size
(9) OMX_SetParameter(handle, OMX_IndexParamPortDefinition, &portDefn) for OUTPUT port
(10) Return OMX_ErrorNone

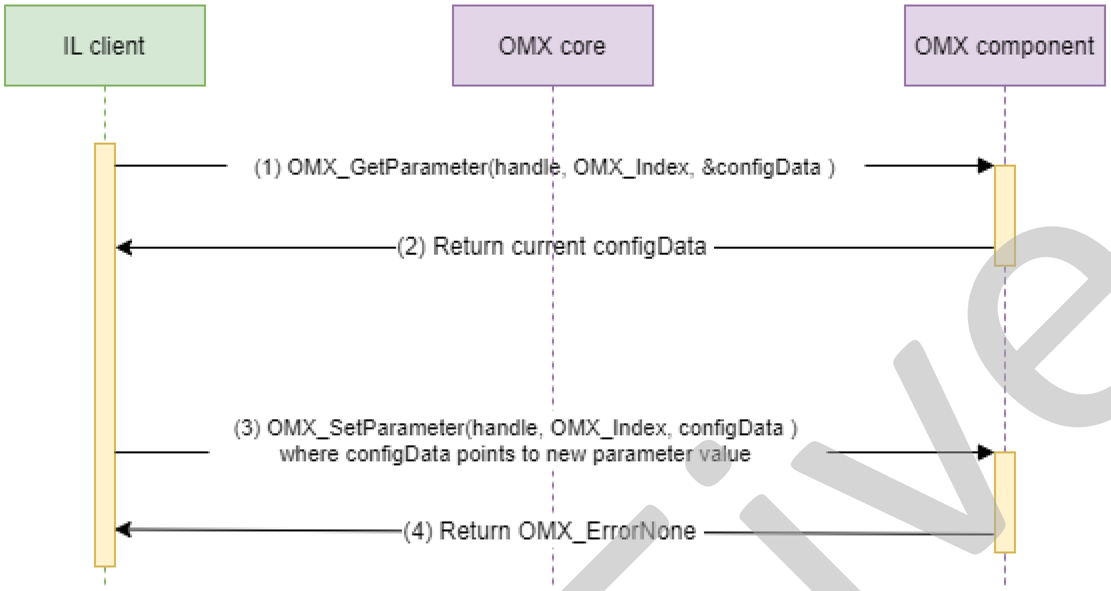
4.3.4. Configure Decoder Parameters

The OMX IL component is able to accept various encode parameters in addition to the mandatory parameters, such as input frame width and height, target frame rate, target bit rate , etc.

1. The IL client can obtain the current configuration using **OMX_GetParameter** as (1) in the following diagram.
2. The IL client must pass the OMX parameter information using **OMX_SetParameter** with the corresponding **OMX_Index** before sending any input data through **OMX_EmptyThisBuffer** as (3) in the following diagram.

The following diagram shows the decoder parameter configuration workflow.

Figure 4-5 Decoder Parameter Configuration Workflow


IL client
OMX core
OMX component
(1) OMX_GetParameter(handle, OMX_Index, &configData)
(2) Return current configData
(3) OMX_SetParameter(handle, OMX_Index, configData)
where configData points to new parameter value
(4) Return OMX_ErrorNone

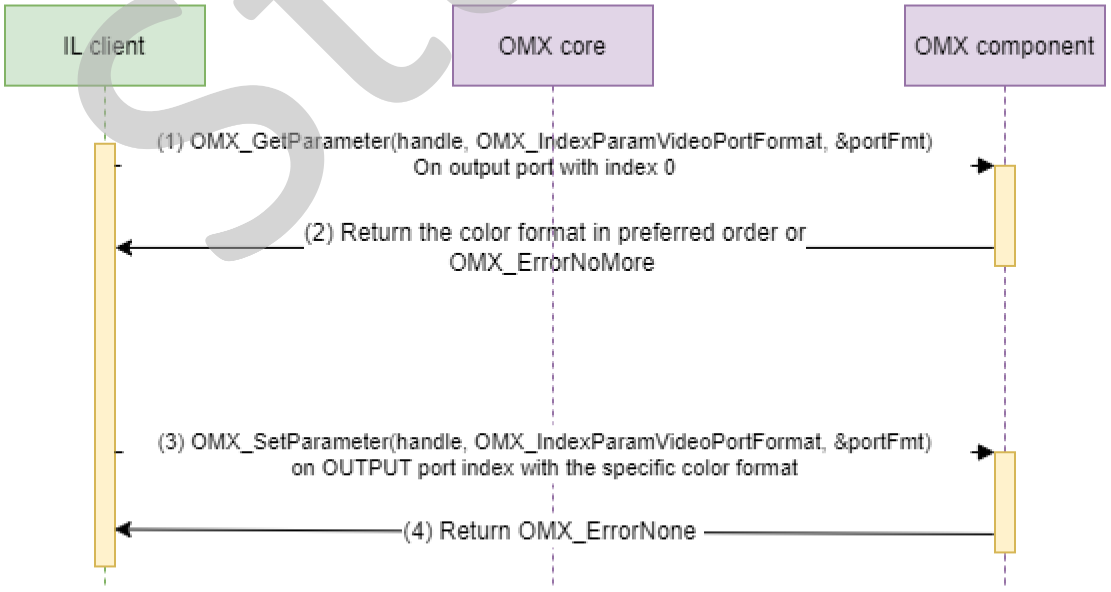
The decoder supports a proprietary color format, requiring an extra step to integrate OMX decoder components.

The decoder supports the **YUV420SemiPlanar** format, that is, YUV planar format, organized with a first plane containing Y pixels and a second plane containing interleaved U and V pixels. U and V pixels are sub-sampled by a factor of two, both horizontally and vertically.

1. The IL client can query the component in the loop shown for supported color formats until the component returns **OMX_ErrorNoMore** as (1) in the following diagram.
2. Index 0 is currently reserved for the **YUV420PackedSemiPlanar32m** color format, which is used to set the color format of the component as (3) in the following diagram.

The following diagram shows the decoder color format configuration workflow.

Figure 4-6 Decoder Color Format Configuration Workflow


IL client
OMX core
OMX component
(1) OMX_GetParameter(handle, OMX_IndexParamVideoPortFormat, &portFmt)
On output port with index 0
(2) Return the color format in preferred order or
OMX_ErrorNoMore
(3) OMX_SetParameter(handle, OMX_IndexParamVideoPortFormat, &portFmt)
on OUTPUT port index with the specific color format
(4) Return OMX_ErrorNone

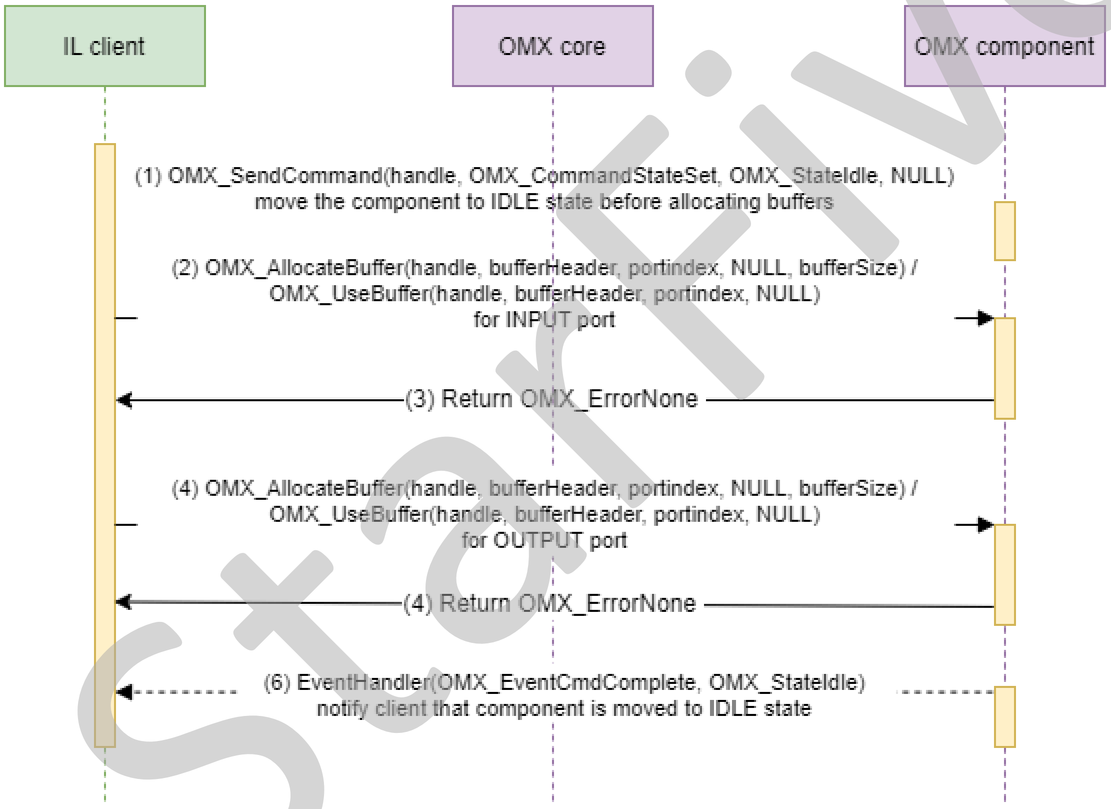
4.3.5. Buffer Allocation

After the component is configured, the next step is to allocate the buffers. Hardware components use physical memory for output buffers to interface with the VPU, so the client uses the **AllocateBuffer** model for the port.

1. To allocate the buffer, the components are moved from the Loaded state to the Idle state as (1) in the following diagram.
2. On the input port, the client can use either **OMX_UseBuffer** or **OMX_AllocateBuffer** calls to the OMX component as (2) in the following diagram. The component is called for the number of input buffers in a loop.
3. On the output port, the client can use either **OMX_UseBuffer** or **OMX_AllocateBuffer** calls to the OMX component as (4) in the following diagram. The component is called for the number of input buffers in a loop
4. Once the buffers are successfully allocated on the input and output ports, the OMX components generate the **OMX_EventCmdComplete** event for the Loaded-to-Idle state transition and send it to the client using **EventHandlerCallback** as (6) in the following diagram.

The following diagram shows the buffer allocation workflow.

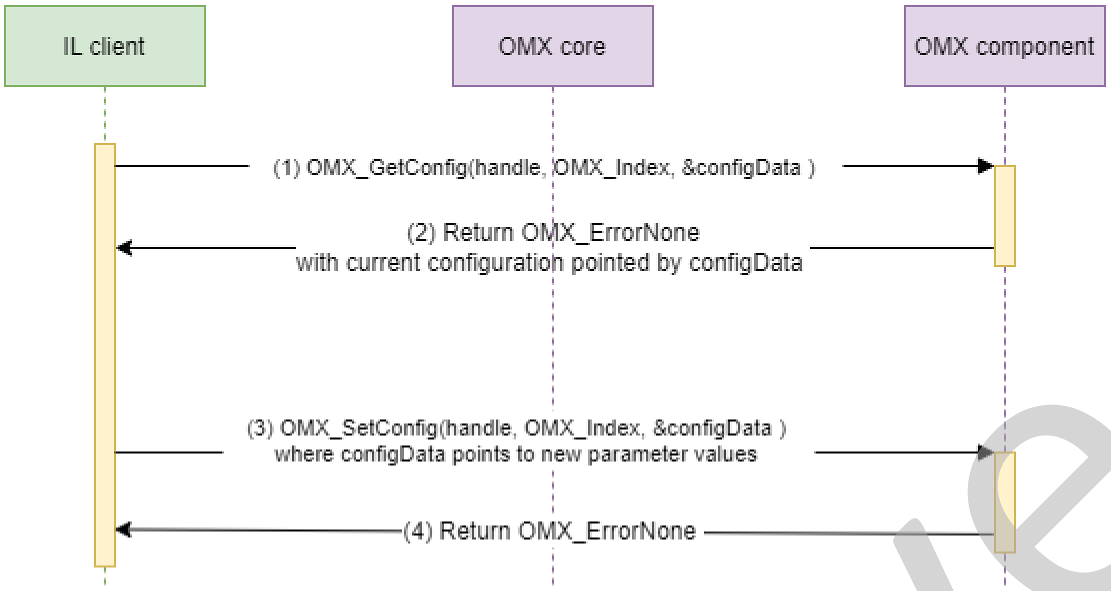
Figure 4-7 Buffer Allocation Workflow


IL client
OMX core
OMX component
(1) OMX_SendCommand(handle, OMX_CommandStateSet, OMX_StateIdle, NULL)
move the component to IDLE state before allocating buffers
(2) OMX_AllocateBuffer(handle, bufferHeader, portindex, NULL, bufferSize) /
OMX_UseBuffer(handle, bufferHeader, portindex, NULL)
for INPUT port
(3) Return OMX_ErrorNone
(4) OMX_AllocateBuffer(handle, bufferHeader, portindex, NULL, bufferSize) /
OMX_UseBuffer(handle, bufferHeader, portindex, NULL)
for OUTPUT port
(4) Return OMX_ErrorNone
(6) EventHandler(OMX_EventCmdComplete, OMX_StateIdle)
notify client that component is moved to IDLE state

4.3.6. Dynamic Port Configuration

The following diagram shows the dynamic port configuration workflow.

Figure 4-8 Dynamic Port Configuration Workflow


IL client
OMX core
OMX component
(1) OMX_GetConfig(handle, OMX_Index, &configData)
(2) Return OMX_ErrorNone
with current configuration pointed by configData
(3) OMX_SetConfig(handle, OMX_Index, &configData)
where configData points to new parameter values
(4) Return OMX_ErrorNone

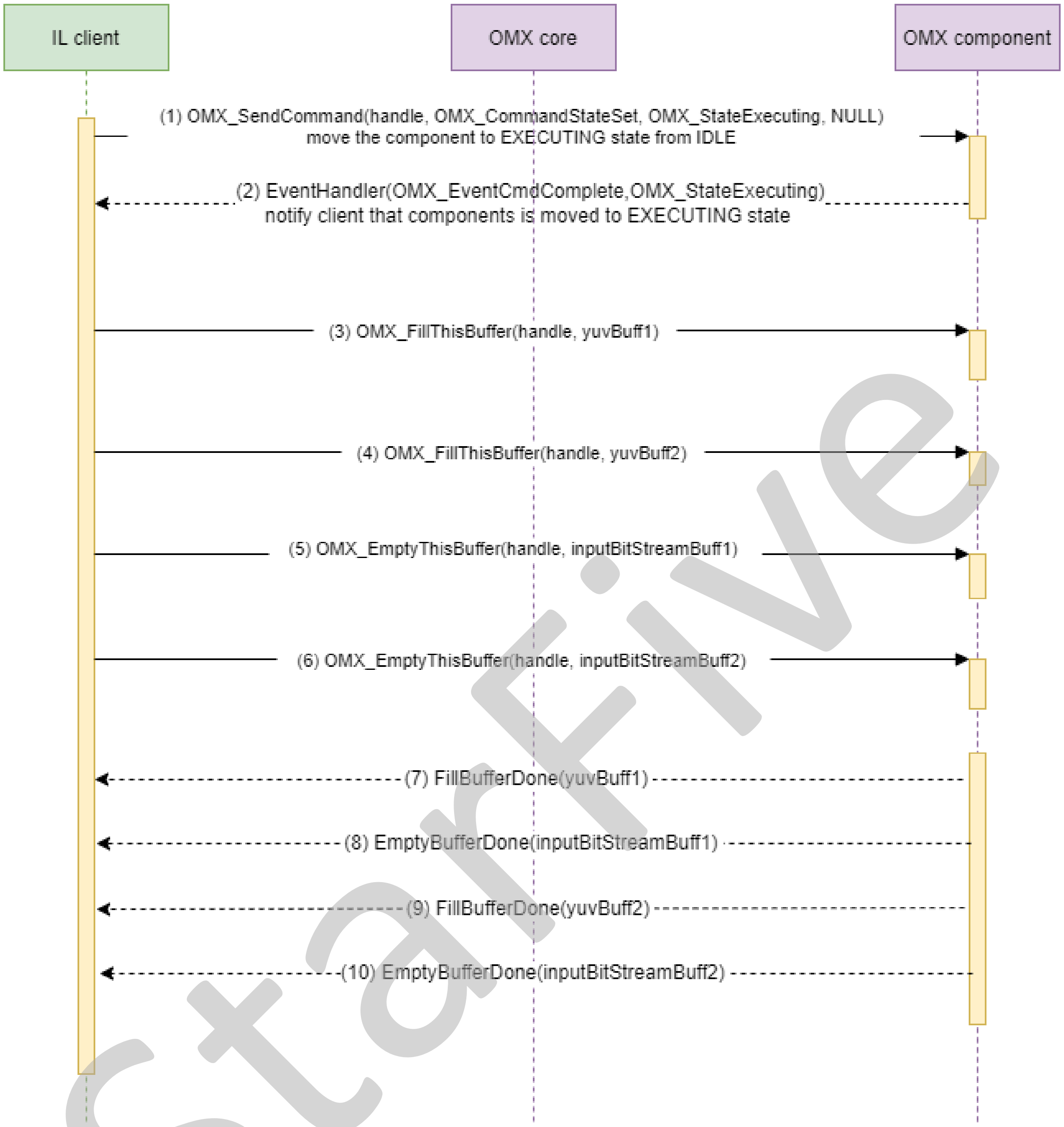
4.3.7. Data Processing

When the component is ready for decoding after buffer allocation and configuration, the client can transition the component to the Executing state, after which the client can start sending the input bitstream for processing.

1. Transition the component from the Idle state to the Executing state as (1) in the following diagram.
2. Wait for **OMX_EventCmdComplete** as (2) in the following diagram.

   Figure 4-9 Data Processing Workflow

3. After the IL client receives the Command Complete event for state transition, it can start sending data to the component.
4. Call **OMX_FillThisBuffer** as (3) and (4) in the following diagram, with the output buffers that can hold the YUV data.
5. Call **OMX_EmptyThisBuffer** as (5) and (6) in the following diagram, with the input bitstream data that is to be decoded.
6. The component generates the **EmptyBufferDone** as (8) and (10) in the following diagram, and **FillBufferDone** as (7) and (9) in the following diagram, callbacks to notify the client after processing the data.


IL client
OMX core
OMX component
(1) OMX_SendCommand(handle, OMX_CommandStateSet, OMX_StateExecuting, NULL)
move the component to EXECUTING state from IDLE
(2) EventHandler(OMX_EventCmdComplete,OMX_StateExecuting)
notify client that components is moved to EXECUTING state
(3) OMX_FillThisBuffer(handle, yuvBuff1)
(4) OMX_FillThisBuffer(handle, yuvBuff2)
(5) OMX_EmptyThisBuffer(handle, inputBitStreamBuff1)
(6) OMX_EmptyThisBuffer(handle, inputBitStreamBuff2)
(7) FillBufferDone(yuvBuff1)
(8) EmptyBufferDone(inputBitStreamBuff1)
(9) FillBufferDone(yuvBuff2)
(10) EmptyBufferDone(inputBitStreamBuff2)

4.3.8. De-Initialization

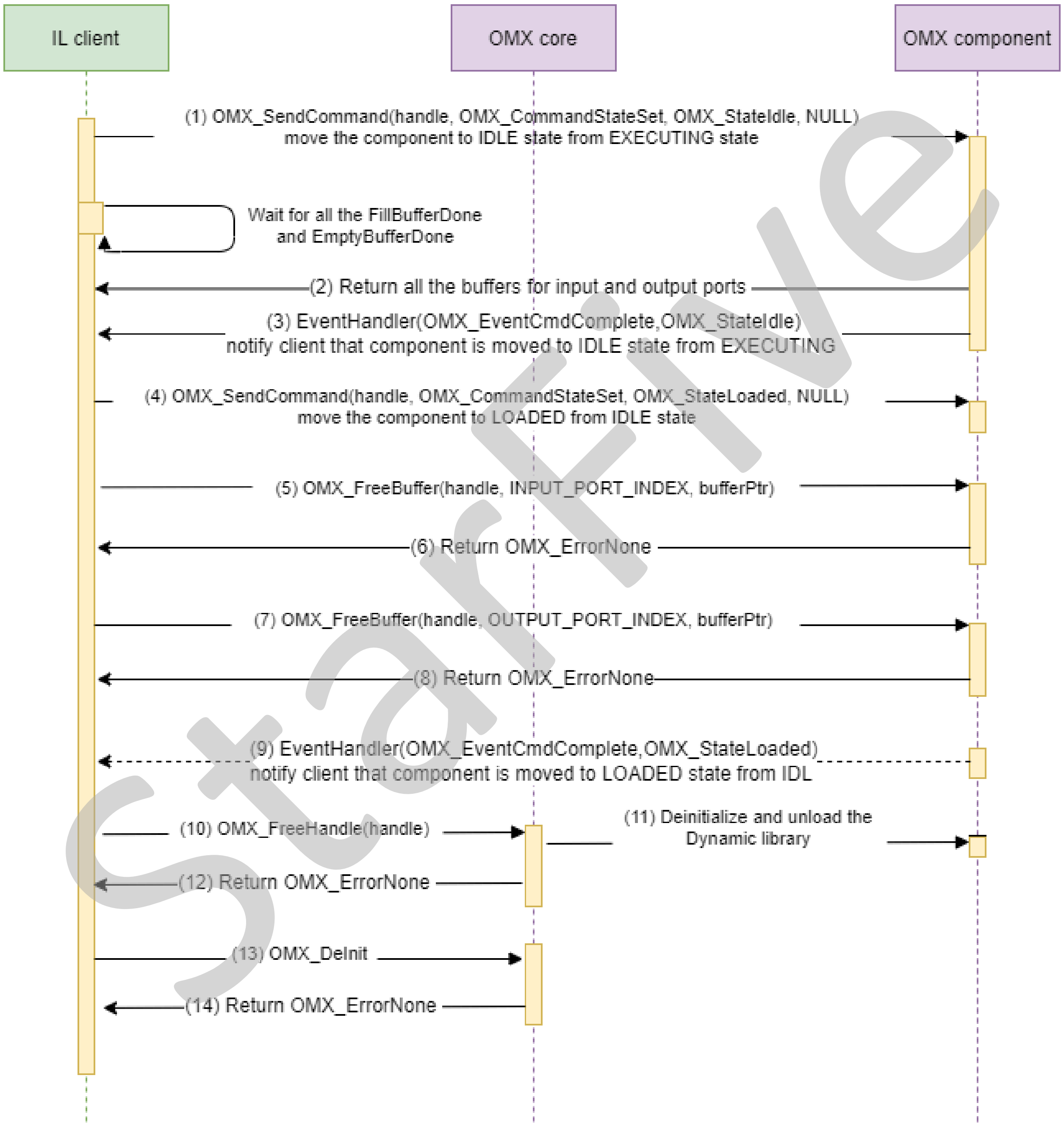
To de-initialize the OpenMAX core, active components must be freed. To free the component, move it to the **Executing→Idle→Loaded** state, and then free all the input and output buffers.

1. Move the component from the **Executing** state to the **Idle** state as (1) in the following diagram.
2. Wait for all buffers to be returned by the component.
3. The component returns all buffers to the **IL client** as (2) in the following diagram.
4. The component generates **OMX_EventCmdComplete** for the **Execute→Idle** state transition as (3) in the following diagram.
5. Transition the component from the **Idle** state to the **Loaded** state as (4) in the following diagram
6. Free all input and output buffers as (5) to (8) in the following diagram.

7. Wait for **OMX_EventCmdComplete** for the **Idle→Loaded** state transition.
8. The client receives the command **OMX_EventCmdComplete** for the **Loaded→Idle** state transition as (9) in the following diagram.
9. Call **OMX_FreeHandle** to the OMX core to release the component handle as (10) in the following diagram.
10. Call **OMX_DeInit** to de-initialize the OMX core as (13) in the following diagram.

The following diagram shows the de-initialization workflow.

Figure 4-10 De-Initialization Workflow


IL client
OMX core
OMX component
(1) OMX_SendCommand(handle, OMX_CommandStateSet, OMX_StateIdle, NULL)
move the component to IDLE state from EXECUTING state
Wait for all the FillBufferDone
and EmptyBufferDone
(2) Return all the buffers for input and output ports
(3) EventHandler(OMX_EventCmdComplete,OMX_StateIdle)
notify client that component is moved to IDLE state from EXECUTING
(4) OMX_SendCommand(handle, OMX_CommandStateSet, OMX_StateLoaded, NULL)
move the component to LOADED from IDLE state
(5) OMX_FreeBuffer(handle, INPUT_PORT_INDEX, bufferPtr)
(6) Return OMX_ErrorNone
(7) OMX_FreeBuffer(handle, OUTPUT_PORT_INDEX, bufferPtr)
(8) Return OMX_ErrorNone
(9) EventHandler(OMX_EventCmdComplete,OMX_StateLoaded)
notify client that component is moved to LOADED state from IDL
(10) OMX_FreeHandle(handle)
(11) Deinitialize and unload the
Dynamic library
(12) Return OMX_ErrorNone
(13) OMX_DeInit
(14) Return OMX_ErrorNone

5. VCODEC

VCODEC is composed of video decoder and video encoder. VCODEC is connected to VCODEC_AHB bus through an AHB slave and VCODEC_AXI through an AXI master.

Video decoder and video encoder share many internal memories and they also share the bus master and slave interfaces. So it prevents video decoder and video encoder from working simultaneously. Encoding and decoding now have to time-share the memory resource on a frame by frame basis.

Video decoder and encoder share the *Memory Management Unit (MMU)* and AXI bus. Video decoder can support cache-able bus operation.

StarFive does not recommend users to use the VCODEC module directly, thus the module relevant user information is currently not opened yet, please contact your StarFive technical support or sales consultant to place a request if you need instructions on how to use this module.

6. VPU

Linux Video Processing Unit allows to offload to the hardware the decoding and encoding of video streams, avoiding the need to involve the CPU for such intensive operations. Having VPU support would provide a smoother video decoding/encoding experience, at a much lower CPU consumption.

The VPU modules support H.264/H.265 video decoding and H.265 video encoding.

StarFive does not recommend users to use the VPU module directly, thus the module relevant user information is currently not opened yet, please contact your StarFive technical support or sales consultant to place a request if you need instructions on how to use this module.