User Manual

8170/8190/9170/9190/G1/G2

HW Decoder OpenMAX IL Component

Version 2.0.6

Copyright Information

This document describes the HW Decoder OpenMAX IL User Manual and remains the official reference source for all revisions/releases of this product until rescinded by an update.

For our current distributors, sales offices, design resource centers, and product information, view our web page located at http://www.verisilicon.com.

For our technical support, email hantro_support@verisilicon.com.

Glossary

1080p	High Definition resolution of 1920x1080 progressive video
API	Application Programming Interface
AVS	Audio Video Coding Standard of China
AVC	Advanced Video Coding, same as H.264
BPP	Bits Per Pixel
DivX	DivX is a brand name of products created by DivX, Inc. The DivX codec uses MPEG-4 compression
EOS	End of Sequence
H.264	Video Coding Standard (ITU-T)
H.263	Video Coding Standard (ITU-T)
HD	High Definition
HEVC	High Efficiency Video Coding, a video coding standard (JCT-VC)
HW	Hardware
IL	OpenMAX Integration Layer
JPEG	Joint Photographic Experts Group, also a common term for still images
MJPEG	Motion JPEG
MPEG-2	Motion Picture Experts Group standard 2 (ISO/IEC 13818-2)
MPEG-4	Motion Picture Experts Group standard 4 (ISO / IEC 14496-2)
NAL	Network Abstraction Layer
OMX	OpenMAX Multimedia interface developed and maintained by the Khronos group
OS	Operating System
OSAL	Operating System Abstraction Layer, a part of OpenMAX conformance testing package
RGB	A color space representation, where red, green and blue light are added together in various ways to reproduce a broad array of colors
RV	Real Video, a video coding specification developed by Real Media Inc.
Semi-planar	A YCbCr storage format, where the luminance samples form one plane in memory, and the pixel by pixel interleaved Cb and Cr samples form another
VC-1	A Microsoft SMPTE 421M video codec
VP6	On2 Truemotion VP6 video coding standard
VP8	Compressed video data format created by On2 Technologies
VP9	Compressed video data format created by Google
WebP	VP8 based still image compression
YCbCr	A color space representation, where color and intensity data are in separate components: Y contains the black and white image (luminance), Cb and Cr the color information (chrominance)
YCbCr 4:2:0	YCbCr sampling format, where Cb and Cr components are sub-sampled by two both horizontally and vertically
YCbCr 4:2:2	YCbCr sampling format, where Cb and Cr components are sub-sampled by two horizontally
YUV	Commonly used alternative for YCbCr

Table of Contents

1 Introduction

This document presents the OpenMAX Integration Layer Application Programming Interface (API) of the Hantro™ 8170/8190/9170/9190/G1/G2 hardware based decoders. The OpenMAX IL API [1] is a standardized multimedia component interface that allows easier and faster integration of multimedia components into a multimedia system. The interface itself is independent of the execution environment and operating system.

The decoders are able to decode H.264 [2], H.263 [3], MPEG-2 [4], MPEG-4 [5], VC-1 [6], RealVideo (supported in 9170/9190/G1), DivX (supported in 9170/9190/G1), VP6 (supported in 9190/G1), AVS [7] (supported in G1), VP8 (supported in G1), HEVC (supported in G2) and VP9 (supported in G2) standards video streams, JPEG standard [8] still images and WebP (supported in G1) still images. The decoders conform to the H.264 Baseline, Main and High profile, H.263 Baseline profile, MPEG-2 Main profile, MPEG-4 Simple and Advanced profile, VC-1 Simple, Main and Advanced profile, RV 8, 9 and 10, DivX 3, 4, 5 and 6, VP6 Simple and Advanced profiles, AVS Jizhun profile, VP8 Main Profile, HEVC Main and Main10 Profile, VP9 Profile 0 and can decode streams up to a maximum picture size of 4k x 4k. The overall performance and capabilities of the decoders is system dependent. Note that formats and features that are presented depend on the configuration that Licensee has chosen for the Decoder Hardware Product and only formats that are applicable to chosen configuration are supported.

The API is developed on Linux environment and the implementation uses an Operating System Abstraction Layer (OSAL) to isolate all OS specific functionality. The Bellagio open-source OMX IL Core for Linux (omxil.sourceforge.net) is used for development and testing purposes but the design allows flexible integration into other OMX Core systems as well.

The usage of the API is described in chapter 3. Chapter 4 gives guidelines to video frame storage formats. The detailed API function interfaces are introduced in chapter 5. References are presented in the end of the document.

In the document all functions, parameters, data types and code are described in `Courier new (syntax style)` font. Notes and filenames are written in *italic* expression.

This document assumes that the reader understands the fundamentals of C-language and the AVS, H.264, H.263, HEVC, MPEG-2, MPEG-4, VC-1, RV, DivX, VP6, VP8, VP9 and JPEG standards.

2 API Version History

Table 1 describes the released API versions, any changes introduced.

TABLE 1. API VERSION HISTORY

API version	Changes/Comments
1.0	Original version according to OMX IL specification v1.1.2
1.1	Hardware configuration limitations explained
1.2	G1 included
1.3	Added AVS format, VP8 and WebP extension support
1.4	Add MJPEG support, minor updates
1.5	Rename custom coding types. Add VSI extension index to table 2.
1.6	Update supported OMX IL indices and extensions. Add supported color formats for JPEG.
2.0	Add G2 decoder support (HEVC and VP9)
2.0.4	Add G2 tiled format and appendix for VSI vendor extension
2.0.6	Update G2 features and VSI vendor extension appendix.

3 Features

The OpenMAX IL API for 8170/8190/9170/9190/G1/G2 decoder is implemented in Linux environment. The Bellagio OMX IL Core for Linux is used for development and testing purposes. The OMX API is implemented on top of the Hantro 8170/8190/9170/9190/G1/G2 decoder AVS, H.264, H.263, HEVC, MPEG-2, MPEG-4, VC-1, RV, DivX, VP6, VP8, VP9, JPEG, WebP and post-processing APIs. The OMX API uses the specific OSAL functions to isolate all OS specific functionality. Figure 1 describes the blocks of the OpenMAX video decoder system.

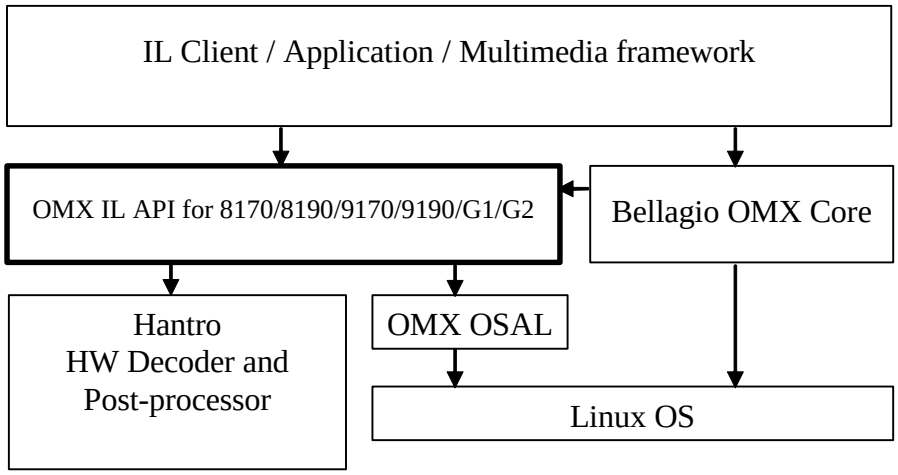

FIGURE 1. OMX API LAYER

Integration to other environments requires the OMX Core and OMX OSAL parts to be implemented for the target environment. The OMX Core handles the component library loading, component creation and tunnel setup as defined in the OMX IL Specification. The OMX OSAL is needed for OS independent implementation of memory allocation, thread and mutex handling.

3.1 OpenMAX IL API Functionality

The OpenMAX IL API is a standardized interface for multimedia components. The interface specifies the functions and their usage for controlling a multimedia object. The data flow from one component to other can be done in three different ways: non-tunneled, tunneled or proprietary communication. The API functions should be non-blocking which implies a threaded implementation of the component. The non-blocking commands must be queued to the component. A set of callback functions is used to inform the IL client (= the user of the IL components) of the component events.

The OMX Core is responsible of loading the component library and creating the component. The decoder API has an initialization function that is called by the OMX Core when creating a component instance.

The 8170/8190/9170/9190/G1/G2 decoder has two input ports and one output port. The first input port format is for compressed AVS, H.264, H.263, HEVC, MJPEG, MPEG-2, MPEG-4, VC-1, RV, DivX, VP6, VP8, VP9, JPEG or WebP stream and the output port format

is uncompressed YUV/RGB video data. For post-processor input the port format is uncompressed YUV video data. The second input port is used for post-processing alpha blending feature and its format is uncompressed YUV or RGB video data.

NOTE: G2 decoder post-processor doesn't support alpha blending.

The decoder control software runs in its own thread. The OMX API functions queue the data to be handled in the decoder thread. The thread processes the commands, handles the decoding and issues callbacks and events to the IL client.

The 8170/8190/9170/9190/G1/G2 decoder supports non-tunneled and tunneled data flow. The component ports are described in Figure 2. Port 0 is used for video/image stream input and port 2 is used for post-processor's alpha blending input. The supported parameters and configurations are listed under the interface functions in Chapter 5.

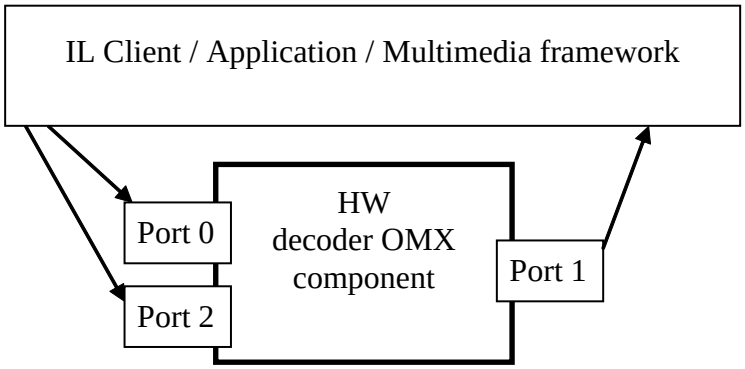

FIGURE 2. OMX COMPONENT PORTS

3.2 OpenMAX IL Features

The 8170/8190/9170/9190/G1/G2 decoder OMX API implementation is done according to OMX IL API version 1.1.2 [1].

The implementation supports OMX IL base profile and tunneling.

The 8170/8190/9170/9190/G1/G2 decoder consists of two OMX components: video decoder component and image decoder component. The OMX names for these components are:

- OMX.hantro.81x0.video.decoder (8170/8190/9170/9190)
- OMX.hantro.81x0.image.decoder (8170/8190/9170/9190)
- OMX.hantro.G1.video.decoder (G1)
- OMX.hantro.G1.image.decoder (G1)
- OMX.hantro.G2.video.decoder (G2)

3.3 Decoder Features

The 8170 decoder has the following features:

- H.264 baseline profile, levels 1-4.1 decoding
    - Byte stream input
    - NAL unit input

- H.263 profile 0, levels 10-70 decoding
- MPEG-2 main profile, low medium and high levels
- MPEG-4 simple and advanced simple profile, levels 0-5 decoding
- VC-1 simple and main profile, levels: low, medium and high, advanced profile, levels 0-3 decoding
    - RCV stream input
- JPEG baseline decoding
- Video post-processing features
    - Frame rotation 90 degrees left/right
    - Frame mirroring horizontally/vertically
    - Frame cropping
    - Frame conversion from YCbCr formats to 16-bit or 32-bit RGB formats
    - Frame scaling with maximum up-scaling factor of 3
    - Two rectangular or alpha blending masks for output frame

The post-processing features can be used in pipeline with the decoder. The post-processing features can also be used as stand-alone, without performing any decoding.

In addition to the 8170 features the 8190 decoder also supports:

- H.264 main and high profile, levels 1-4.1 decoding

In addition to the 8190 features the 9170 decoder has the following features:

- RV 8, 9 and 10 decoding
- DivX 3, 4, 5 and 6 decoding

In addition to the 9170 features the 9190 decoder has the following features:

- VP6.0, 6.1, 6.2, Simple and Advanced Profiles decoding

In addition to the 9190 features the G1 decoder has the following features:

- H.264 main and high profile, levels 1-5.1 decoding
- VP8 decoding
- AVS Jizhun profile decoding
- WebP decoding

The G2 decoder has the following features:

- HEVC Main and Main10 Profile decoding
- VP9 Profile 0 and Profile 2 decoding
- Tiled to raster scan conversion
- Downscaling **not supported**

3.4 Input Buffers

For video frame input buffers the decoders support following compression formats:

- OMX_VIDEO_CodingAVC (supported in 8190/9170/9190/G1)
- OMX_VIDEO_CodingMPEG2 (supported in 8190/9170/9190/G1)
- OMX_VIDEO_CodingMPEG4 (supported in 8190/9170/9190/G1)
- OMX_VIDEO_CodingH263 (supported in 8190/9170/9190/G1)
- OMX_VIDEO_CodingWMV (supported in 8190/9170/9190/G1)
- OMX_VIDEO_CodingSORENSON (supported in 8190/9170/9190/G1)
- OMX_VIDEO_CodingRV (supported in 9170/9190/G1)
- OMX_VIDEO_CodingDIVX (supported in 9170/9190/G1)
- OMX_VIDEO_CodingDIVX3 (supported in 9170/9190/G1)
- OMX_VIDEO_CodingVP6 (supported in 9190/G1)

- OMX_VIDEO_CodingAVS (supported in G1)
- OMX_VIDEO_CodingVP8 (supported in G1)
- OMX_VIDEO_CodingMJPEG (supported in 8190/9170/9190/G1)
- OMX_VIDEO_CodingHEVC (supported in G2)
- OMX_VIDEO_CodingVP9 (supported in G2)
- OMX_IMAGE_CodingJPEG (supported in 8190/9170/9190/G1)
- OMX_IMAGE_CodingWebP (supported in G1)

For post-processor stand-alone mode (8170/8190/9170/9190/G1) input the decoder support following color formats:

- OMX_COLOR_FormatYUV420PackedPlanar
- OMX_COLOR_FormatYUV420PackedSemiPlanar
- OMX_COLOR_FormatYCbYCr
- OMX_COLOR_FormatYCrYCb (supported in 8190/9170/9190/G1)
- OMX_COLOR_FormatCbYCrY (supported in 8190/9170/9190/G1)
- OMX_COLOR_FormatCrYCbY (supported in 8190/9170/9190/G1)

For post-processor alpha blending input (port 2) decoder supports following color formats:

- OMX_COLOR_Format32bitARGB8888

The decoder supports following three payload formats as documented in OMX IL Specification chapter 2.1.12:

1. Full buffer that may consist of several complete decoding units/frames and the last decoding unit/frame may be partial
2. Several complete decoding units/frames per buffer
3. One complete decoding unit/frame per buffer

For H.264 NAL unit stream, VC-1, VP8 and VP9 stream only payload format 3 is supported. This is due to the fact that those streams don’t contain start codes that identify the start of a decoding unit.

For HEVC stream, only payload format 3 is supported due the limitation of G2 decoder.

The input buffer format and size indicate which payload format is used. For compressed input formats, when the input buffer is completely filled it is assumed to be of payload format 1. For uncompressed input formats, when the buffer size is not a multiple of frame size it is assumed to be of payload format 1.

When using payload format 1 with compressed input it is required that the decoder searches the buffer for start code prefix and uses a temporary buffer to store partial decoding units. Searching the start code prefix and copying the stream to temporary buffer may degrade the decoding performance and thus it is recommended to use the input buffer formats 2 and 3 whenever possible. Typically the stream storage or transport format stores the size of each individual decoding unit.

It is recommended that the component is allowed to allocate all its input buffers. This can be done by using the OMX API function *AllocateBuffer.* The component will allocate buffers that are HW accessible, thus removing any need to copy data from input buffer to internal buffer. Using *UseBuffer* instead may degrade decoding performance.

When choosing the size of the input buffer it is recommended to use a size large enough to fit the biggest decoding unit, or the size of one complete uncompressed input frame.

3.5 Output Buffers

When initializing the decoder it is impossible to know the dimensions of the output frame. The decoding begins with the stream headers and once they are decoded the decoder dispatches a PortSettingsChanged-event to indicate that the output port dimensions are known.

When the client receives the PortSettingsChanged-event it should first disable the output port of the decoder. Then it should query the size of the output buffer from the decoder's output port definition by using the *GetParameter* function. Finally the client should enable the output port and re-allocate the output buffers with the correct buffer size.

For output data the following color formats are supported:

- OMX_COLOR_FormatYUV420PackedSemiPlanar

The following color formats are also supported if post processor (8170/8190/9170/9190/G1) is enabled in the hardware and OpenMAX IL component configuration:

- OMX_COLOR_FormatYCbYCr
- OMX_COLOR_FormatYCrYCb (supported in 8190/9170/9190/G1)
- OMX_COLOR_FormatCbYCrY (supported in 8190/9170/9190/G1)
- OMX_COLOR_FormatCrYCbY (supported in 8190/9170/9190/G1)
- OMX_COLOR_Format16bitARGB1555
- OMX_COLOR_Format16bitARGB4444
- OMX_COLOR_Format16bitRGB565
- OMX_COLOR_Format16bitBGR565
- OMX_COLOR_Format25bitARGB1888
- OMX_COLOR_Format32bitARGB8888
- OMX_COLOR_Format32bitBGRA8888
- OMX_COLOR_FormatYUV422PackedSemiPlanar (For 4:2:2 encoded JPEG only)
- OMX_COLOR_FormatL8 (JPEG only)
- OMX_COLOR_FormatYUV411PackedSemiPlanar (JPEG only)
- OMX_COLOR_FormatYUV440PackedSemiPlanar (JPEG only)
- OMX_COLOR_FormatYUV444PackedSemiPlanar (JPEG only)

For output buffers the only supported payload format is one complete frame per buffer. This means that the decoder stores each output frame in separate output buffer.

Using 8170/8190/9170/9190/G1 without post processor will degrade decoding performance because output data is copied from internal buffer to OMX component output buffer. G2 decoder doesn't have this limitation.

G2 post processor supports also following color formats:

- OMX_COLOR_FormatYUV420SemiPlanar4x4Tiled
- OMX_COLOR_FormatYUV420SemiPlanarP010 (when bit depth is 10)

4 Video Frame Storage Format

This chapter describes the different input and output picture storage formats supported by the decoder and post-processor. Notice that some of the formats are only used for input or output.

4.1 YCbCr 4:2:0 Planar Format

In the planar format each video sample component forms one memory plane. The luminance and both chrominance planes must be stored in a linear and contiguous memory block as shown in Figure 3. The luminance samples are stored in raster-scan order ($Y_0Y_1Y_2Y_3Y_4$….). The chrominance samples are stored in two planes also in raster scan order ($Cb_0Cb_1Cb_2Cb_3$… and $Cr_0Cr_1Cr_2Cr_3$…).

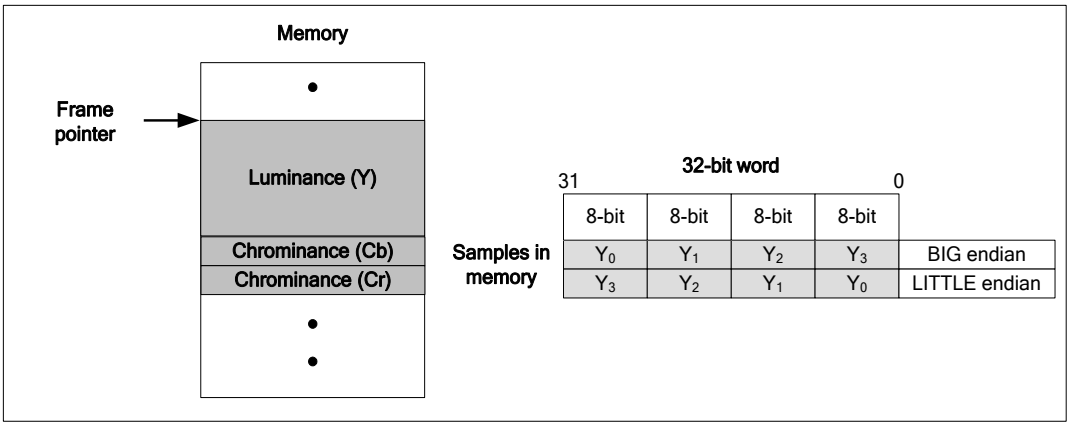

FIGURE 3. YCBCR 4:2:0 PLANAR FORMAT EXTERNAL MEMORY USAGE.

The YCbCr 4:2:0 planar format is supported only as an input format for the post-processor. In this format each pixel takes 12 bits of memory.

4.2 YCbCr 4:2:0 Semi-Planar Format

In semi-planar YCbCr 4:2:0 format the luminance samples form one plane in memory, and chrominance samples form another. The luminance and chrominance planes must be stored in a linear and contiguous memory block as presented in Figure 4. The luminance pixels are stored in raster-scan order $Y_0Y_1Y_2Y_3Y_4$…. The interleaved chrominance CbCr samples are stored in raster-scan order in memory as $Cb_0Cr_0Cb_1Cr_1Cb_2Cr_2Cb_3$....

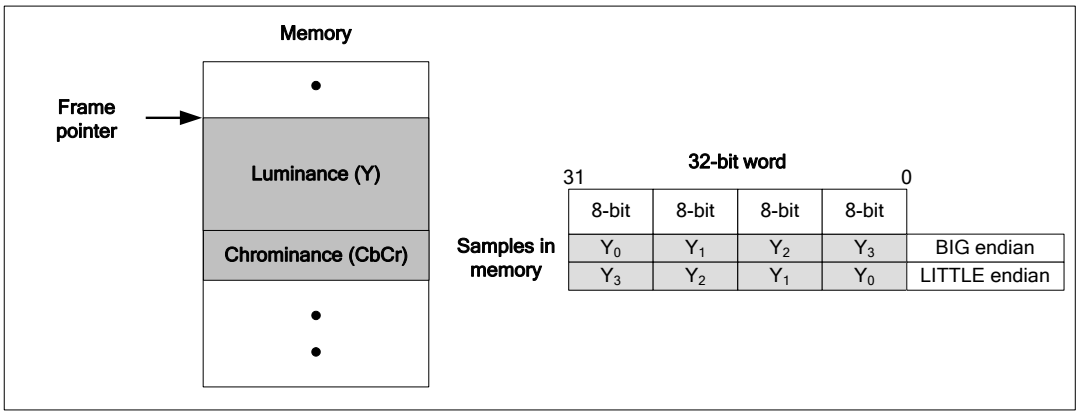

FIGURE 4. YCBCR 4:2:0 SEMI-PLANAR FORMAT EXTERNAL MEMORY USAGE.

The YCbCr 4:2:0 semi-planar format is supported both as an input and as an output format for the post-processor. In this format each pixel takes 12 bits of memory. As the chrominance components are interleaved, the bus load caused by this format can be slightly lower than with the planar format due to the reduced amount of non-sequential memory addressing.

4.3 YCbCr 4:2:2 Interleaved Format

In the interleaved YCbCr 4:2:2 format the pixel samples form a single plane in which the data has to be stored linearly and contiguously as shown in Figure 5. The pixel data is in raster scan order and the chrominance samples are interleaved between the luminance samples as $Y_0Cb_0Y_1Cr_0Y_2Cb_1Y_3Cr_1$….

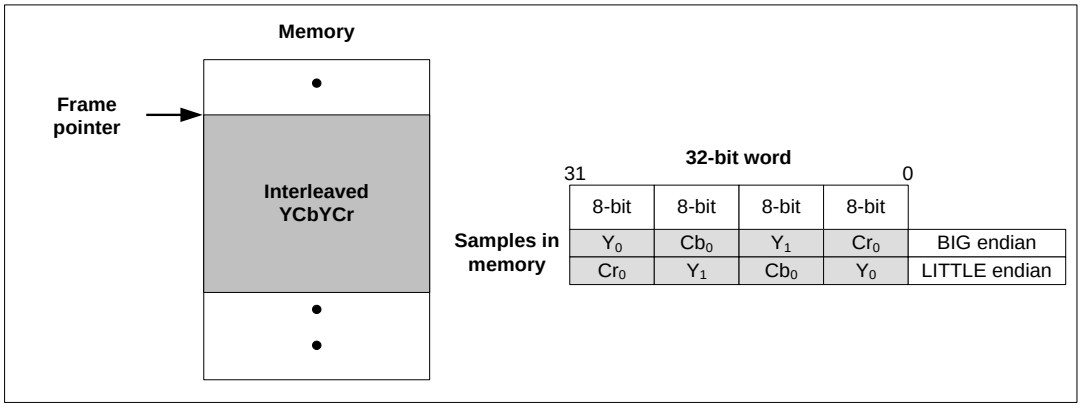

FIGURE 5. YCBYCR 4:2:2 INTERLEAVED FORMAT EXTERNAL MEMORY USAGE.

The YCbCr 4:2:2 interleaved format is supported both as an input and as an output format for the post-processor. In this format each pixel takes 16 bits of memory. Although there is more data to be transferred than in the 4:2:0 formats, the interleaved format is bus effective as there are no non-sequential memory accesses caused by the plane changes.

4.4 G2 Decoder Tiled 4x4 Format

The output picture of the decoder is in semi-planar YCbCr 4:2:0 4x4 tiled format, i.e. luminance data forms one plane in memory, and chrominance data forms another. The distinction from raster scan format is that the output picture is grouped into tiles, which are then stored linearly and contiguously in the memory. The chrominance tiles have to be stored right after the luminance tiles in external memory as shown in Figure 6.

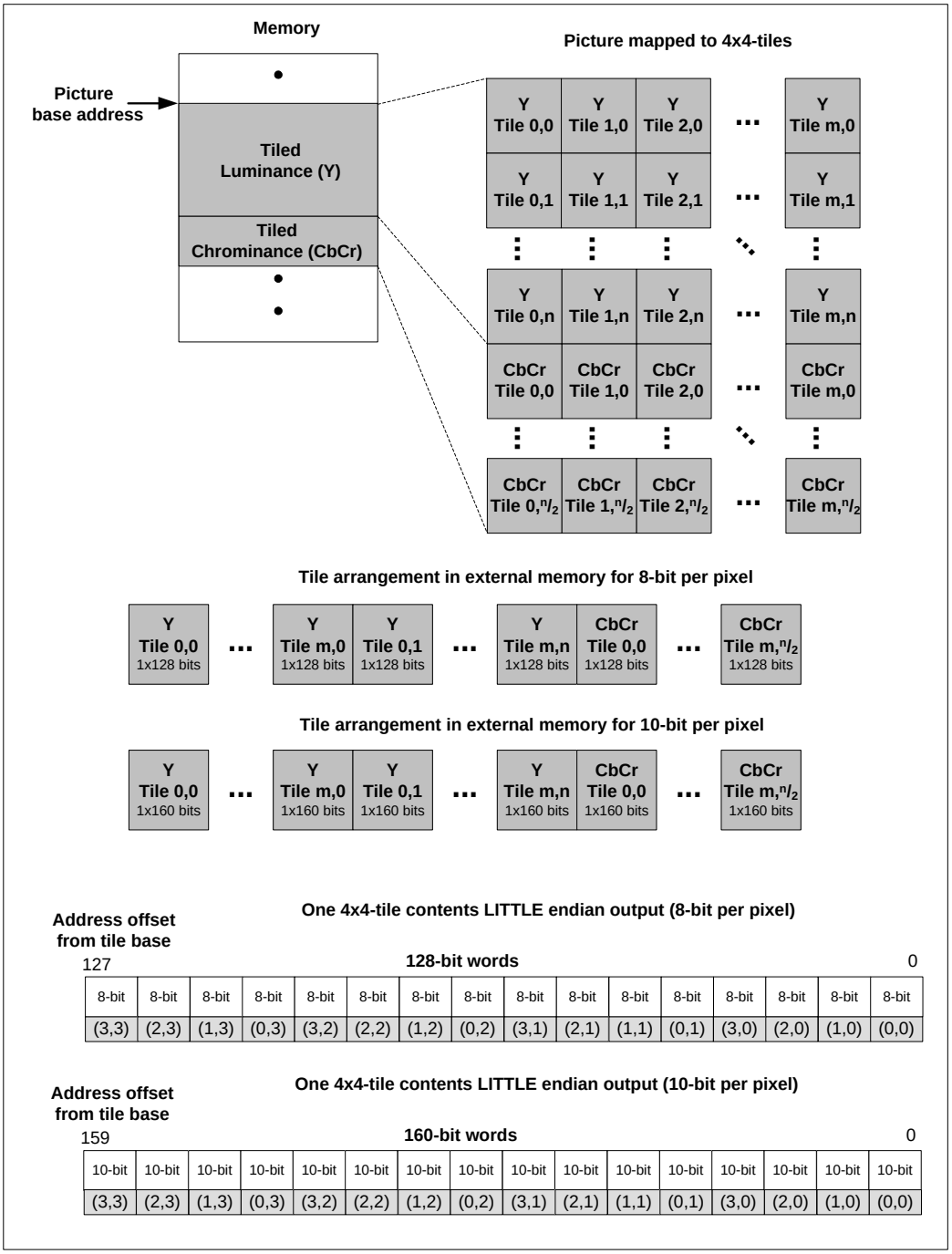

FIGURE 6. TILED 4X4 FORMAT EXTERNAL MEMORY USAGE

4.5 RGB 16bpp Format

In this format each pixel is represented by 16 or less bits containing the red, blue and green samples. There are several 16bpp formats which use different number of bits for each sample. For example the RGB 5-5-5 format uses 5 bits for each sample and 1 bit is left unused or can represent a transparency flag, where RGB 5-6-5 uses 6 bits for the G sample and 5 bits for R and B samples. Common for all 16bpp types is that two pixels fit into one 32-bit space.

The data has to be stored linearly and contiguously in the memory as shown in Figure 7. The order of the sample bytes is always defined with a 16 bit word which will define the byte order in the memory. The pixel samples are stored in raster-scan order.

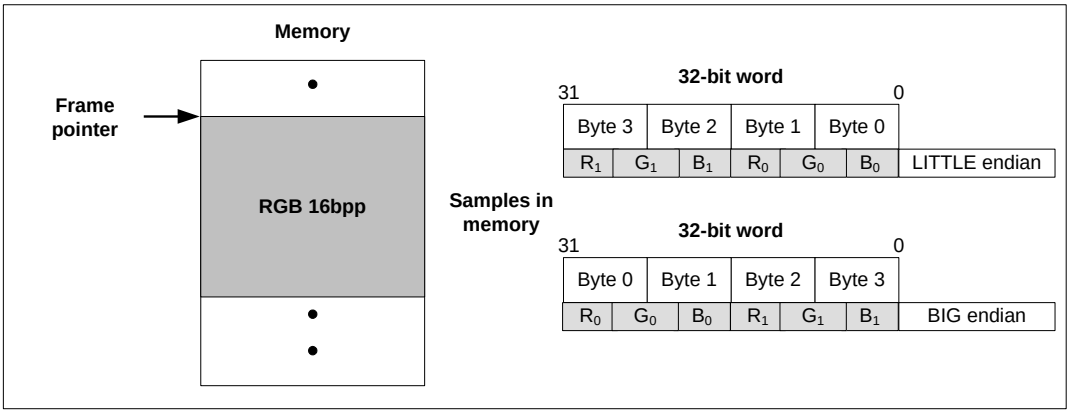

FIGURE 7. RGB 16BPP FORMAT EXTERNAL MEMORY USAGE.

The RGB 16bpp formats are supported only as post-processor output formats.

4.6 RGB 32bpp Format

Any RGB format that has its pixels represented by more than 16 bits each is considered to be of 32bpp type. Typically in this format each pixel is represented by three bytes containing a red, blue and green sample and a 4th byte which can be empty or hold an alpha blending value. Common for all 32bpp types is that only one pixel fit into one 32-bit space.

The data has to be stored linearly and contiguously in the memory as shown in Figure 8. The order of the sample bytes is always defined on a 32 bit word and that will define the byte order in the memory (for example, ARGB order means that on a little endian system the B sample will be stored in the lowest offset byte followed by the G, R and A samples. In a big endian system the A sample will be on the lowest offset byte). The pixel samples are stored in raster-scan order.

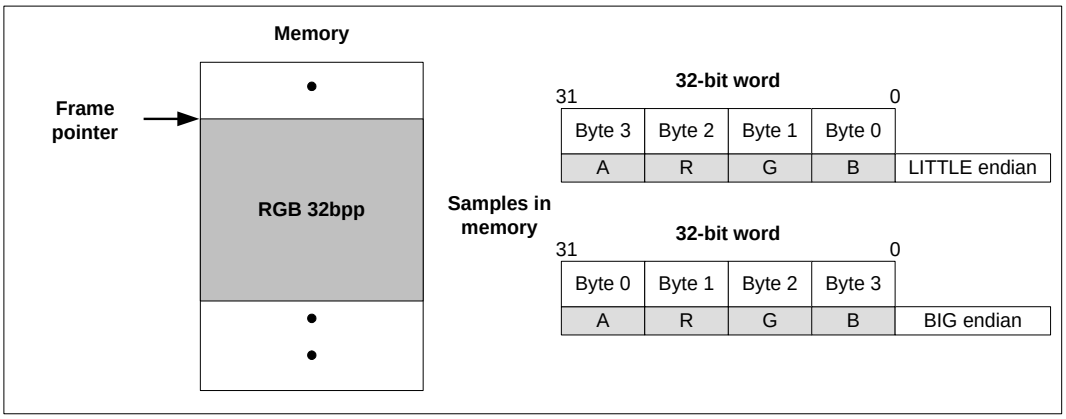

FIGURE 8. RGB 32BPP FORMAT EXTERNAL MEMORY USAGE.

5 Interface Functions

This chapter describes the interface functions and parameters of the OpenMAX IL API as implemented for the 8170/8190/9170/9190/G1/G2 decoders. The complete documentation of the OMX API functions and their usage can be found in the OpenMAX IL Specification [1]. The Initialization function is called from the OMX Core when creating a component instance and after that the OMX defined macros (for example *OMX_GetComponentVersion*) should be used for invoking all the other functions.

5.1 HantroHwDecOmx_video_constructor

Syntax

```
OMX_ERRORTYPE HantroHwDecOmx_video_constructor(
            OMX_OUT OMX_COMPONENTTYPE* comp,
            OMX_IN  OMX_STRING name)
```

Purpose

This function will allocate and initialize the private structures of a component instance. The handle pointing to OMX_COMPONENTTYPE structure should be allocated by the OMX Core before calling this function.

5.2 HantroHwDecOmx_image_constructor

Syntax

```
OMX_ERRORTYPE HantroHwDecOmx_image_constructor(
            OMX_OUT OMX_COMPONENTTYPE* comp,
            OMX_IN  OMX_STRING name)
```

Purpose

This function will allocate and initialize the private structures of a component instance. The handle pointing to OMX_COMPONENTTYPE structure should be allocated by the OMX Core before calling this function.

5.3 GetComponentVersion

Syntax

```
OMX_ERRORTYPE GetComponentVersion(
            OMX_IN  OMX_HANDLETYPE hComponent,
            OMX_OUT OMX_STRING pComponentName,
            OMX_OUT OMX_VERSIONTYPE* pComponentVersion,
            OMX_OUT OMX_VERSIONTYPE* pSpecVersion,
            OMX_OUT OMX_UUIDTYPE* pComponentUUID)
```

Purpose

This function will return version information about the component.

The decoder will implement the referenced IL Specification version [1].

5.4 SendCommand

Syntax

```
OMX_ERRORTYPE SendCommand(
            OMX_IN  OMX_HANDLETYPE hComponent,
            OMX_IN  OMX_COMMANDTYPE Cmd,
            OMX_IN  OMX_U32 nParam1,
            OMX_IN  OMX_PTR pCmdData)
```

Purpose

Send a command to the component. This call is a non-blocking call. The component should check the parameters and then queue the command to the component thread to be executed. The component thread shall send the EventHandler() callback at the conclusion of the command.

Description

Following commands are supported:

- OMX_CommandStateSet
  - OMX_StateLoaded
  - OMX_StateIdle
  - OMX_StateExecuting
  - OMX_StatePause
  - OMX_StateInvalid
- OMX_CommandFlush
- OMX_CommandPortDisable
- OMX_CommandPortEnable
- OMX_CommandMarkBuffer

5.5 GetParameter

Syntax

```
OMX_ERRORTYPE GetParameter(
            OMX_IN  OMX_HANDLETYPE hComponent,
            OMX_IN  OMX_INDEXTYPE nParamIndex,
            OMX_INOUT OMX_PTR ComponentParameterStructure)
```

Purpose

The function will get one of the current parameter settings from the component. The nParamIndex parameter is used to indicate which structure is being requested from the component. The application shall allocate the correct structure and shall fill in the structure size and version information before invoking this macro. When the parameter

applies to a port, the caller shall fill in the appropriate nPortIndex value indicating the port on which the parameter applies. If the component has not had any settings changed, then the component should return a set of valid DEFAULT parameters for the component.

Some parameters may be disabled if not supported by the decoder HW, see Chapter 3 for more information.

Description

TABLE 2. SUPPORTED PARAMETER INDEXES FOR GETPARAMETER IN VIDEO DOMAIN

Parameter index, port and parameter name	Parameter structure and description
OMX_IndexParamVideoInit	OMX_PORT_PARAM_TYPE
OMX_IndexParamCompBufferSupplier	OMX_PARAM_BUFFERSUPPLIERTYPE
OMX_IndexParamPortDefinition	OMX_PARAM_PORTDEFINITIONTYPE
OMX_IndexParamVideoPortFormat	OMX_VIDEO_PARAM_PORTFORMATTYPE
OMX_IndexParamVideoAvc	OMX_VIDEO_PARAM_AVCTYPE
OMX_IndexParamVideoMpeg4	OMX_VIDEO_PARAM_MPEG4TYPE
OMX_IndexParamVideoMpeg2	OMX_VIDEO_PARAM_MPEG2TYPE
OMX_IndexParamVideoH263	OMX_VIDEO_PARAM_H263TYPE
OMX_IndexParamVideoWmv	OMX_VIDEO_PARAM_WMVTYPE
OMX_IndexParamVideoRv	OMX_VIDEO_PARAM_RVTYPE
OMX_IndexParamVp8	OMX_VIDEO_PARAM_VP8TYPE
OMX_IndexParamVideoHevc	OMX_VIDEO_PARAM_HEVCTYPE VSI vendor extension for HEVC format
OMX_IndexParamVideoVp9	OMX_VIDEO_PARAM_VP9TYPE VSI vendor extension for VP9 format
OMX_IndexParamVideoProfileLevelCurrent	OMX_VIDEO_PARAM_PROFILELEVELTYPE
OMX_IndexParamVideoProfileLevelQuerySupported	OMX_VIDEO_PARAM_PROFILELEVELTYPE
OMX_IndexParamCommonDeblocking	OMX_PARAM_DEBLOCKINGTYPE
OMX_IndexParamStandardComponentRole	OMX_PARAM_COMPONENTROLETYPE
OMX_IndexParamPriorityMgmt	OMX_PRIORITYMGMTTYPE
OMX_IndexParamVideoMvcStream	OMX_VIDEO_PARAM_MVCSTREAMTYPE VSI vendor extension for configuring H.264 MVC mode
OMX_IndexParamVideoG2Config	OMX_VIDEO_PARAM_G2CONFIGTYPE VSI vendor extension for configuring G2 decoder

TABLE 3. SUPPORTED PARAMETER INDEXES FOR GETPARAMETER IN IMAGE DOMAIN

Parameter index, port and parameter name	Parameter structure and description
OMX_IndexParamImageInit	OMX_PORT_PARAM_TYPE
OMX_IndexParamCompBufferSupplier	OMX_PARAM_BUFFERSUPPLIERTYPE
OMX_IndexParamPortDefinition	OMX_PARAM_PORTDEFINITIONTYPE
OMX_IndexParamImagePortFormat	OMX_IMAGE_PARAM_PORTFORMATTYPE
OMX_IndexParamStandardComponentRole	OMX_PARAM_COMPONENTROLETYPE
OMX_IndexParamPriorityMgmt	OMX_PRIORITYMGMTTYPE

5.6 SetParameter

Syntax

```
OMX_ERRORTYPE SetParameter(
            OMX_IN  OMX_HANDLETYPE hComponent,
            OMX_IN  OMX_INDEXTYPE nIndex,
            OMX_IN  OMX_PTR ComponentParameterStructure)
```

Purpose

The function will send an initialization parameter structure to a component. Each structure shall be sent one at a time, in a separate invocation of the macro. This macro can only be invoked when the component is in the OMX_LoadedState state, or the port is disabled (when the parameter applies to a port). The nParamIndex parameter is used to indicate which structure is being passed to the component. The application shall allocate the correct structure and shall fill in the structure size and version information (as well as the actual data) before invoking this macro. The application is free to dispose of this structure after the call as the component is required to copy any data it shall retain.

Some parameters may be disabled if not supported by the decoder HW, see Chapter 3 for more information.

Description

TABLE 4. SUPPORTED PARAMETER INDEXES FOR SETPARAMETER IN VIDEO DOMAIN

Parameter index, port and parameter name		Parameter structure and description
OMX_IndexParamVideoInit		OMX_PORT_PARAM_TYPE
OMX_IndexParamCompBufferSupplier		OMX_PARAM_BUFFERSUPPLIERTYPE
OMX_IndexParamPortDefinition		OMX_PARAM_PORTDEFINITIONTYPE
port=0	bFlagErrorConcealment	Enable/disable decoder error concealment
	eCompressionFormat	Defines the input stream type
	nFrameWidth, nFrameHeight	Defines the post-processor input frame resolution
	eColorFormat	Defines the post-processor input frame format
		OMX_COLOR_FormatYUV420PackedPlanar
		OMX_COLOR_FormatYUV420PackedSemiPlanar
		OMX_COLOR_FormatYCbYCr
		OMX_COLOR_FormatYCrYCb
		OMX_COLOR_FormatCbYCrY
		OMX_COLOR_FormatCrYCbY
port=1	nFrameWidth, nFrameHeight	Defines the post-processor output frame resolution
	eColorFormat	OMX_COLOR_FormatYUV420PackedSemiPlanar
		OMX_COLOR_FormatYCbYCr
		OMX_COLOR_FormatCbYCrY
		OMX_COLOR_FormatCrYCbY
		OMX_COLOR_FormatCrYCbY
		OMX_COLOR_Format16bitARGB1555
		OMX_COLOR_Format16bitARGB4444
		OMX_COLOR_Format16bitRGB565
		OMX_COLOR_Format16bitBGR565
		OMX_COLOR_Format25bitARGB1888
		OMX_COLOR_Format32bitARGB8888
		OMX_COLOR_Format32bitBGRA8888
port=2	nFrameWidth, nFrameHeight	Defines the alpha-blending mask input resolution
OMX_IndexParamVideoPortFormat		OMX_VIDEO_PARAM_PORTFORMATTYPE
OMX_IndexParamVideoAvc		OMX_VIDEO_PARAM_AVCTYPE
OMX_IndexParamVideoMpeg4		OMX_VIDEO_PARAM_MPEG4TYPE
OMX_IndexParamVideoMpeg2		OMX_VIDEO_PARAM_MPEG2TYPE
OMX_IndexParamVideoH263		OMX_VIDEO_PARAM_H263TYPE
OMX_IndexParamVideoWmv		OMX_VIDEO_PARAM_WMVTYPE
OMX_IndexParamVideoRv		OMX_VIDEO_PARAM_RVTYPE
OMX_IndexParamVp8		OMX_VIDEO_PARAM_VP8TYPE
OMX_IndexParamCommonDeblocking		OMX_PARAM_DEBLOCKINGTYPE
port=0	bDeblocking	Enable/disable MPEG-4/H.263 deblocking filter
OMX_IndexParamCommonDithering		OMX_PARAM_DITHERTYPE
OMX_IndexParamStandardComponentRole		OMX_PARAM_COMPONENTROLETYPE
OMX_IndexParamPriorityMgmt		OMX_PRIORITYMGMTTYPE
OMX_IndexParamVideoMvcStream		OMX_VIDEO_PARAM_MVCSTREAMTYPE VSI vendor extension for configuring H.264 MVC mode
OMX_IndexParamVideoG2Config		OMX_VIDEO_PARAM_G2CONFIGTYPE VSI vendor extension for configuring G2 decoder

TABLE 5. SUPPORTED PARAMETER INDEXES FOR SETPARAMETER IN IMAGE DOMAIN

Parameter index, port and parameter name		Parameter structure and description
OMX_IndexParamImageInit		OMX_PORT_PARAM_TYPE
OMX_IndexParamCompBufferSupplier		OMX_PARAM_BUFFERSUPPLIERTYPE
OMX_IndexParamPortDefinition		OMX_PARAM_PORTDEFINITIONTYPE
port=0	nFrameWidth, nFrameHeight	Defines the post-processor input frame resolution
	eColorFormat	Defines the post-processor input frame format
		OMX_COLOR_FormatYUV420PackedPlanar
		OMX_COLOR_FormatYUV420PackedSemiPlanar
		OMX_COLOR_FormatYCbYCr
		OMX_COLOR_FormatCbYCrY
		OMX_COLOR_FormatCrYCbY
		OMX_COLOR_FormatCrYCbY
port=1	nFrameWidth, nFrameHeight	Defines the post-processor output frame resolution
	eColorFormat	OMX_COLOR_FormatYUV420PackedSemiPlanar
		OMX_COLOR_FormatYCbYCr
		OMX_COLOR_FormatCbYCrY
		OMX_COLOR_FormatCrYCbY
		OMX_COLOR_FormatCrYCbY
		OMX_COLOR_Format16bitARGB1555
		OMX_COLOR_Format16bitARGB4444
		OMX_COLOR_Format16bitRGB565
		OMX_COLOR_Format16bitBGR565
		OMX_COLOR_Format25bitARGB1888
		OMX_COLOR_Format32bitARGB8888
		OMX_COLOR_Format32bitBGRA8888
		OMX_COLOR_FormatL8
		OMX_COLOR_FormatYUV422PackedSemiPlanar
		OMX_COLOR_FormatYUV411PackedSemiPlanar
		OMX_COLOR_FormatYUV440PackedSemiPlanar
		OMX_COLOR_FormatYUV444PackedSemiPlanar
port=2	nFrameWidth, nFrameHeight	Defines the alpha-blending mask input resolution
OMX_IndexParamCommonDithering		OMX_PARAM_DITHERTYPE
OMX_IndexParamImagePortFormat		OMX_IMAGE_PARAM_PORTFORMATTYPE
OMX_IndexParamStandardComponentRole		OMX_PARAM_COMPONENTROLETYPE
OMX_IndexParamPriorityMgmt		OMX_PRIORITYMGMTTYPE

5.7 GetConfig

Syntax

```
OMX_ERRORTYPE GetConfig(
              OMX_IN  OMX_HANDLETYPE hComponent,
              OMX_IN  OMX_INDEXTYPE nIndex,
              OMX_INOUT OMX_PTR pParam)
```

Purpose

The function will get one of the configuration structures from a component. This function can be invoked any time after the component has been loaded. The nParamIndex call parameter is used to indicate which structure is being requested from the component. The application shall allocate the correct structure and shall fill in the structure size and version information before invoking this function. If the component has not had this configuration parameter sent before, then the component should return a set of valid DEFAULT values for the component.

NOTE: G2 decoder post processor doesn’t support features below.

Description

TABLE 6. SUPPORTED PARAMETER INDEXES FOR GETCONFIG IN VIDEO DOMAIN

Parameter index, port and parameter name	Parameter structure and description
OMX_IndexConfigCommonRotate	OMX_CONFIG_ROTATIONTYPE
OMX_IndexConfigCommonMirror	OMX_CONFIG_MIRRORTYPE
OMX_IndexConfigCommonDithering	OMX_CONFIG_DITHERTYPE
OMX_IndexConfigCommonInputCrop	OMX_CONFIG_RECTTYPE
OMX_IndexConfigCommonContrast	OMX_CONFIG_CONTRASTTYPE
OMX_IndexConfigCommonBightness	OMX_CONFIG_BRIGHTNESSTYPE
OMX_IndexConfigCommonSaturation	OMX_CONFIG_SATURATIONTYPE
OMX_IndexConfigCommonPlaneBlend	OMX_CONFIG_PLANEBLENDTYPE
OMX_IndexConfigCommonOutputPosition	OMX_CONFIG_POINTTYPE
OMX_IndexConfigCommonExclusionRect	OMX_CONFIG_RECTTYPE
OMX_IndexConfigCommonOutputCrop	OMX_CONFIG_RECTTYPE
OMX_IndexConfigVideoVp8ReferenceFrameType	OMX_VIDEO_VP8REFERENCEFRAMEINFOTYPE

TABLE 7. SUPPORTED PARAMETER INDEXES FOR GETCONFIG IN IMAGE DOMAIN

Parameter index	Parameter structure
OMX_IndexConfigCommonRotate	OMX_CONFIG_ROTATIONTYPE
OMX_IndexConfigCommonMirror	OMX_CONFIG_MIRRORTYPE
OMX_IndexConfigCommonDithering	OMX_CONFIG_DITHERTYPE
OMX_IndexConfigCommonInputCrop	OMX_CONFIG_RECTTYPE
OMX_IndexConfigCommonContrast	OMX_CONFIG_CONTRASTTYPE
OMX_IndexConfigCommonBrightness	OMX_CONFIG_BRIGHTNESSTYPE
OMX_IndexConfigCommonSaturation	OMX_CONFIG_SATURATIONTYPE
OMX_IndexConfigCommonPlaneBlend	OMX_CONFIG_PLANEBLENDTYPE
OMX_IndexConfigCommonOutputPosition	OMX_CONFIG_POINTTYPE
OMX_IndexConfigCommonExclusionRect	OMX_CONFIG_RECTTYPE

5.8 SetConfig

Syntax

```
OMX_ERRORTYPE SetConfig(
              OMX_IN  OMX_HANDLETYPE hComponent,
              OMX_IN  OMX_INDEXTYPE nIndex,
              OMX_IN  OMX_PTR pParam)
```

Purpose

The function will send one of the configuration structures to a component. Each structure shall be sent one at a time, each in a separate invocation of the function. This macro can be invoked any time after the component has been loaded. The application shall allocate the correct structure and shall fill in the structure size and version information (as well as the actual data) before invoking this macro. The application is free to dispose of this structure after the call as the component is required to copy any data it shall retain.

NOTE: G2 decoder post processor doesn’t support features below.

Description

TABLE 8. SUPPORTED PARAMETER INDEXES FOR SETCONFIG IN VIDEO DOMAIN

Parameter index		Parameter structure
OMX_IndexConfigCommonRotate		OMX_CONFIG_ROTATIONTYPE
port=1	nRotation	Defines the output frame rotation +- 90 degrees
OMX_IndexConfigCommonMirror		OMX_CONFIG_MIRRORTYPE
port=1	eMirror	Defines the output frame horizontal/vertical mirroring
OMX_IndexConfigCommonInputCrop		OMX_CONFIG_RECTTYPE
port=1	nTop	Defines the amount of cropped pixels on top edge of the input frame
	nLeft	Defines the amount of cropped pixels on left edge of the input frame
	nWidth	Defines the cropped frame horizontal resolution
	nHeight	Defines the cropped frame vertical resolution
OMX_IndexConfigCommonContrast		OMX_CONFIG_CONTRASTTYPE
port=1	nContrast	Defines the post-processing contrast adjustment
OMX_IndexConfigCommonBrightness		OMX_CONFIG_BRIGHTNESSTYPE
port=1	nBrightness	Defines the post-processing brightness adjustment
OMX_IndexConfigCommonPlaneBlend		OMX_CONFIG_PLANEBLENDTYPE
port=1	nAlpha	Defines the alpha value for output RGB frame
OMX_IndexConfigCommonDithering		OMX_CONFIG_DITHERTYPE
port=1	eDither	Enable/Disable dithering
OMX_IndexConfigCommonSaturation		OMX_CONFIG_SATURATIONTYPE
port=1	nSaturation	Defines the post-processing saturation adjustment
OMX_IndexConfigCommonExclusionRect		OMX_CONFIG_RECTTYPE
port=1	nLeft, nTop, nWidth, nHeight	Defines the output mask position and dimensions
OMX_IndexConfigCommonOutputPosition		OMX_CONFIG_POINTTYPE
port=2	nX, nY	Defines the alpha-blending mask position

TABLE 9. SUPPORTED PARAMETER INDEXES FOR SETCONFIG IN IMAGE DOMAIN

Parameter index		Parameter structure
OMX_IndexConfigCommonRotate		OMX_CONFIG_ROTATIONTYPE
port=1	nRotation	Defines the input frame rotation +- 90 degrees
OMX_IndexConfigCommonMirror		OMX_CONFIG_MIRRORTYPE
port=1	eMirror	Defines the output frame horizontal/vertical mirroring
OMX_IndexConfigCommonInputCrop		OMX_CONFIG_RECTTYPE
port=1	nTop	Defines the amount of cropped pixels on top edge of the input frame
	nLeft	Defines the amount of cropped pixels on left edge of the input frame
	nWidth	Defines the cropped frame horizontal resolution
	nHeight	Defines the cropped frame vertical resolution
OMX_IndexConfigCommonContrast		OMX_CONFIG_CONTRASTTYPE
port=1	nContrast	Defines the post-processing contrast adjustment
OMX_IndexConfigCommonBrightness		OMX_CONFIG_BRIGHTNESSTYPE
port=1	nBrightness	Defines the post-processing brightness adjustment
OMX_IndexConfigCommonPlaneBlend		OMX_CONFIG_PLANEBLENDTYPE
port=1	nAlpha	Defines the alpha value for output RGB frame
OMX_IndexConfigCommonDithering		OMX_CONFIG_DITHERTYPE
port=1	eDither	Enable/Disable dithering
OMX_IndexConfigCommonSaturation		OMX_CONFIG_SATURATIONTYPE
port=1	nSaturation	Defines the post-processing saturation adjustment
OMX_IndexConfigCommonExclusionRect		OMX_CONFIG_RECTTYPE
port=1	nLeft, nTop, nWidth, nHeight	Defines the output mask position and dimensions
OMX_IndexConfigCommonOutputPosition		OMX_CONFIG_POINTTYPE
port=2	nX, nY	Defines the alpha-blending mask position

5.9 GetExtensionIndex

Syntax

```
OMX_ERRORTYPE GetExtensionIndex(
            OMX_IN  OMX_HANDLETYPE hComponent,
            OMX_IN  OMX_STRING cParameterName,
            OMX_OUT OMX_INDEXTYPE* pIndexType)
```

Purpose

The function will invoke a component to translate a vendor specific configuration or parameter string into an OMX structure index. There is no requirement for the vendor to support this command for the indexes already found in the OMX_INDEXTYPE enumeration (this is done to save space in small components). The component shall support all vendor supplied extension indexes not found in the master OMX_INDEXTYPE enumeration.

TABLE 10. SUPPORTED EXTENSION INDICES IN VIDEO DOMAIN

Parameter index	Parameter structure
OMX.google.android.index.enableAndroidNativeBuffers	EnableAndroidNativeBuffersParams
OMX.google.android.index.getAndroidNativeBufferUsage	GetAndroidNativeBufferUsageParams
OMX.google.android.index.useAndroidNativeBuffer2	UseAndroidNativeBuffersParams

NOTE: User is responsible for low level implementation of Android native buffer allocation.

5.10 GetState

Syntax

```
OMX_ERRORTYPE GetState(
            OMX_IN  OMX_HANDLETYPE hComponent,
            OMX_OUT OMX_STATETYPE* pState)
```

Purpose

The function will invoke the component to get the current state of the component and place the state value into the location pointed to by pState.

5.11 ComponentTunnelRequest

Syntax

```
OMX_ERRORTYPE ComponentTunnelRequest(
              OMX_IN  OMX_HANDLETYPE hComponent,
              OMX_IN  OMX_U32 nPort,
              OMX_IN  OMX_HANDLETYPE hTunneledPort,
              OMX_IN  OMX_U32 nTunneledPort,
              OMX_INOUT  OMX_TUNNELSETUPTYPE* pTunnelSetup)
```

Purpose

The function will interact with another OMX component to determine if tunneling is possible and to setup the tunneling. The return codes for this method can be used to determine if tunneling is not possible, or if tunneling is not supported.

When this method is invoked when nPort in an output port, the component will:

1. Populate the pTunnelSetup structure with the output port's requirements and constraints for the tunnel.

When this method is invoked when nPort in an input port, the component will:

1. Query the necessary parameters from the output port to determine if the ports are compatible for tunneling
2. If the ports are compatible, the component should store the tunnel step provided by the output port
3. Determine which port (either input or output) is the buffer supplier, and call OMX_SetParameter on the output port to indicate this selection.

5.12 UseBuffer

Syntax

```
OMX_ERRORTYPE UseBuffer(
            OMX_IN OMX_HANDLETYPE hComponent,
            OMX_INOUT OMX_BUFFERHEADERTYPE** ppBuffer,
            OMX_IN OMX_U32 nPortIndex,
            OMX_IN OMX_PTR pAppPrivate,
            OMX_IN OMX_U32 nSizeBytes,
            OMX_IN OMX_U8* pBuffer)
```

Purpose

The function will request that the component use a buffer (and allocate its own buffer header) already allocated by another component, or by the IL Client.

For description of the buffer payload formats and buffer sizes see Chapter 3.

5.13 AllocateBuffer

Syntax

```
OMX_ERRORTYPE AllocateBuffer(
            OMX_IN OMX_HANDLETYPE hComponent,
            OMX_INOUT OMX_BUFFERHEADERTYPE** ppBuffer,
            OMX_IN OMX_U32 nPortIndex,
            OMX_IN OMX_PTR pAppPrivate,
            OMX_IN OMX_U32 nSizeBytes)
```

Purpose

The function will request that the component allocate a new buffer and buffer header. The component will allocate the buffer and the buffer header and return a pointer to the buffer header.

For description of the buffer payload formats and buffer sizes see Chapter 3.

5.14 FreeBuffer

Syntax

```
OMX_ERRORTYPE FreeBuffer(
            OMX_IN  OMX_HANDLETYPE hComponent,
            OMX_IN  OMX_U32 nPortIndex,
            OMX_IN  OMX_BUFFERHEADERTYPE* pBufferHeader)
```

Purpose

The function will release a buffer header from the component which was allocated using either OMX_AllocateBuffer or OMX_UseBuffer. If the component allocated the buffer (see OMX_UseBuffer) then the component shall free the buffer and buffer header.

5.15 FillThisBuffer

Syntax

```
OMX_ERRORTYPE FillThisBuffer(
            OMX_IN  OMX_HANDLETYPE hComponent,
            OMX_IN  OMX_BUFFERHEADERTYPE* pBufferHeader)
```

Purpose

The function will send an empty buffer to an output port of a component. The buffer will be filled by the component and returned to the application via the FillBufferDone call back. This is a non-blocking call in that the component will record the buffer and return immediately and then fill the buffer, later, at the proper time. As expected, this macro may be invoked only while the component is in the OMX_ExecutingState. If nPortIndex does not specify an output port, the component shall return an error.

5.16 EmptyThisBuffer

Syntax

```
OMX_ERRORTYPE EmptyThisBuffer(
            OMX_IN  OMX_HANDLETYPE hComponent,
            OMX_IN  OMX_BUFFERHEADERTYPE* pBufferHeader)
```

Purpose

The function will send a buffer full of data to an input port of a component. The buffer will be emptied by the component and returned to the application via the EmptyBufferDone call back. This is a non-blocking call in that the component will record the buffer and return immediately and then empty the buffer, later, at the proper time. As expected, this macro may be invoked only while the component is in the OMX_ExecutingState. If nPortIndex does not specify an input port, the component shall return an error.

5.17 SetCallbacks

Syntax

```
OMX_ERRORTYPE SetCallbacks(
            OMX_IN  OMX_HANDLETYPE hComponent,
            OMX_IN  OMX_CALLBACKTYPE* pCallbacks,
            OMX_IN  OMX_PTR pAppData)
```

Purpose

The function is used by the core to specify the callback structure from the application to the component.

5.18 ComponentDeInit

Syntax

```
OMX_ERRORTYPE ComponentDeInit(
            OMX_IN  OMX_HANDLETYPE hComponent)
```

Purpose

The function is used to de-initialize the component providing a means to free any resources allocated at component initialization. After this call the component handle is not valid for further use.

6 Usage

This chapter demonstrates the basic usage of the 8170/8190/9170/9190/G1/G2 decoder OMX component. A thorough documentation of the OpenMAX API usage is in OMX IL API Specification [1].

6.1 Component creation

Figure 9 describes the process of creating a component instance and allocating buffers. Detailed description of this process can be found in the OMX IL Specification.

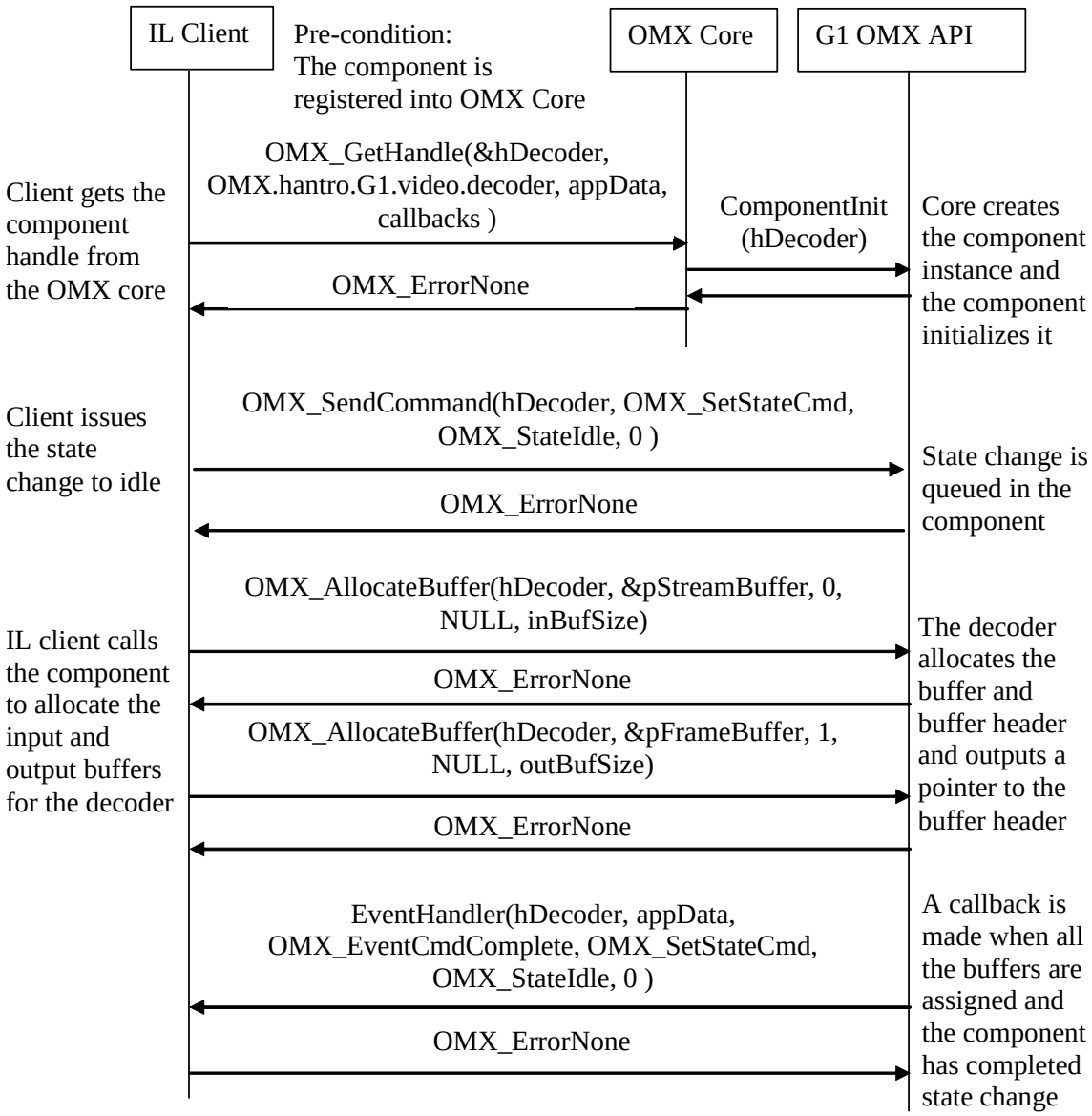

FIGURE 9. OMX API USAGE FOR COMPONENT CREATION

6.2 Decoding

Figure 10 describes the usage of the component when decoding a frame with non-tunneled data flow. The component creation is done as described above.

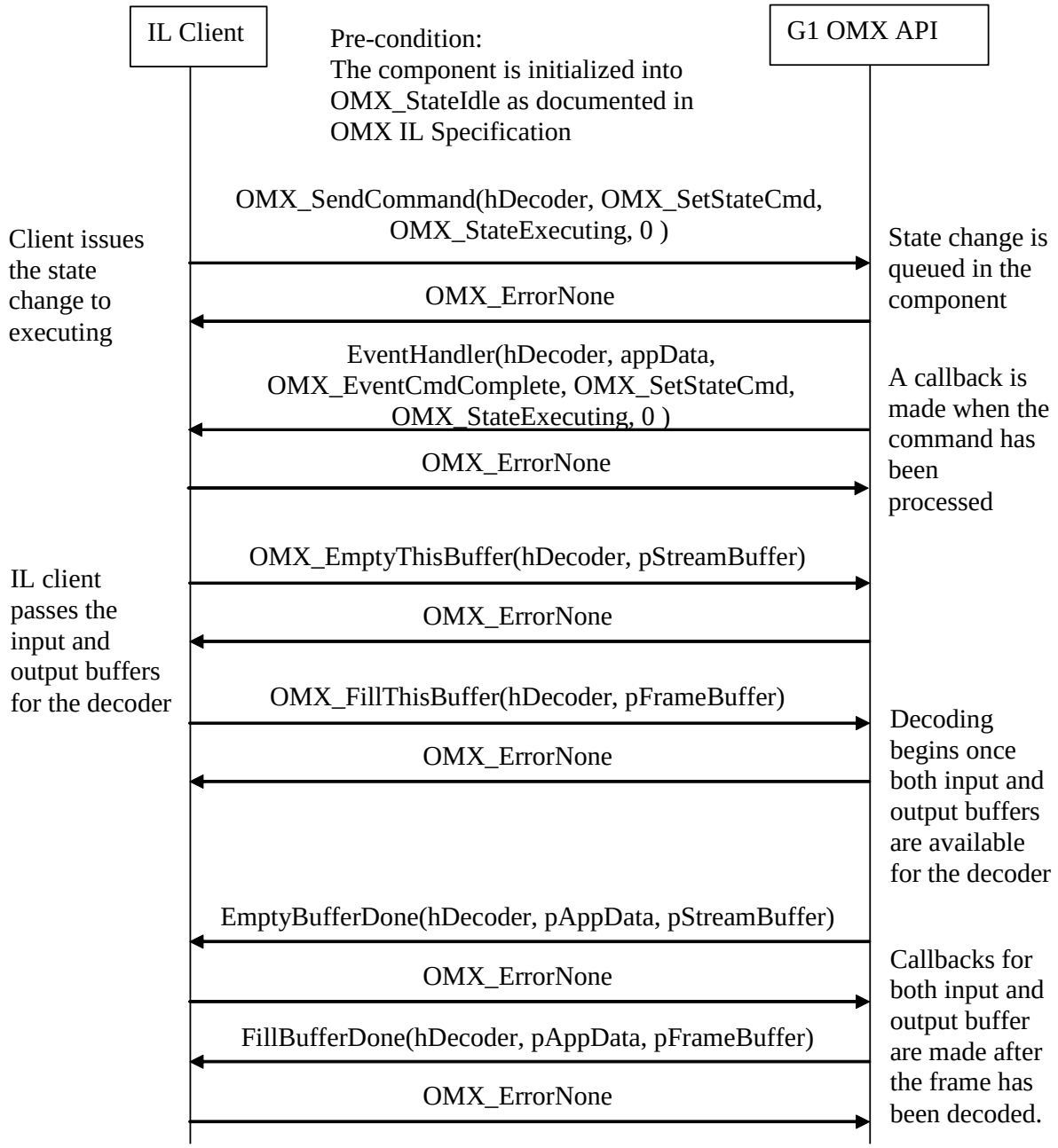

FIGURE 10. OMX API USAGE FOR DECODING A FRAME

References

[1] OpenMAX IL API Specification v1.1.2, The Khronos Group, 2008.
[2] ISO/IEC FDIS 14496-10: Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding.
[3] ITU-T Recommendation H.263 (02/98), Video coding for low bit rate communication.
[4] INTERNATIONAL STANDARD ISO/IEC 13818-2, Information technology - Generic coding of moving pictures and associated audio information: Video.
[5] ISO/IEC 14496-2: Information technology – Coding of audio-visual objects – Part 2: Visual, Third edition 2004-06-01.
[6] SMPTE Standard for Television: VC-1 Compressed Video Bitstream Format and Decoding Process, SMPTE 421M (2006).
[7] The Standards of People's Republic of China, Information technology – Advanced coding of audio and video – Part 2: Video
[8] ISO/IEC IS 10918-1: Information technology – Digital compression and coding of continuous-tone still images, Oct 20, 1999.

Appendix

VSI extension for OpenMAX IL Specification 1.1.2

1.1 Buffer Flags

1.1.1 Second View Frame Flag

This flag is set when the buffer content contains second view frame from MVC stream

```
#define OMX_BUFFERFLAG_SECOND_VIEW      0x00010000
```

1.1.2 VP8 Temporal Layer Frame Flags

One of these flags is set when the buffer contains encoded VP8 temporal layer frame

```
#define OMX_BUFFERFLAG_BASE_LAYER       0x00020000
#define OMX_BUFFERFLAG_FIRST_LAYER      0x00040000
#define OMX_BUFFERFLAG_SECOND_LAYER     0x00080000
#define OMX_BUFFERFLAG_THIRD_LAYER      0x00100000
```

1.2 Enumerations

1.2.1 OMX_INDEXVSITYPE

Extends OMX_INDEXTYPE defined in the OpenMAX IL API 1.1.2

```
typedef enum OMX_INDEXVSITYPE {
    OMX_IndexVsiStartUnused = OMX_IndexVendorStartUnused + 0x00100000,
    OMX_IndexParamVideoMvcStream,
    OMX_IndexConfigVideoIntraArea,
    OMX_IndexConfigVideoRoiArea,
    OMX_IndexConfigVideoRoiDeltaQp,
    OMX_IndexConfigVideoAdaptiveRoi,
    OMX_IndexConfigVideoVp8TemporalLayers,
    OMX_IndexParamVideoHevc,
    OMX_IndexParamVideoVp9
} OMX_INDEXVSITYPE;
```

1.2.2 OMX_VIDEO_CODINGVSITYPE

Extends OMX_VIDEO_CODINGTYPE defined in the OpenMAX IL API 1.1.2

```
typedef enum OMX_VIDEO_CODINGVSITYPE {
    OMX_VIDEO_CodingVsiStartUnused   =   OMX_VIDEO_CodingVendorStartUnused   +
0x00100000,
    OMX_VIDEO_CodingSORENSON,
    OMX_VIDEO_CodingDIVX,
    OMX_VIDEO_CodingDIVX3,
    OMX_VIDEO_CodingVP6,
    OMX_VIDEO_CodingAVS,
```

```
    OMX_VIDEO_CodingHEVC,
    OMX_VIDEO_CodingVP9
} OMX_VIDEO_CODINGVSITYPE;
```

1.2.3 OMX_COLOR_FORMATVSITYPE

Extends OMX_COLOR_FORMATTYPE defined in the OpenMAX IL API 1.1.2

```
typedef enum OMX_COLOR_FORMATVSITYPE {
    OMX_COLOR_FormatVsiStartUnused  =  OMX_COLOR_FormatVendorStartUnused  +
0x00100000,
    OMX_COLOR_FormatYUV411SemiPlanar,
    OMX_COLOR_FormatYUV411PackedSemiPlanar,
    OMX_COLOR_FormatYUV440SemiPlanar,
    OMX_COLOR_FormatYUV440PackedSemiPlanar,
    OMX_COLOR_FormatYUV444SemiPlanar,
    OMX_COLOR_FormatYUV444PackedSemiPlanar
} OMX_COLOR_FORMATVSITYPE;
```

1.2.4 OMX_VIDEO_HEVCPROFILETYPE

Defines HEVC/H.265 video profile types

```
typedef enum OMX_VIDEO_HEVCPROFILETYPE {
    OMX_VIDEO_HEVCProfileMain     = 0x01,   /**< Main profile */
    OMX_VIDEO_HEVCProfileMain10   = 0x02,   /**< Main10 profile */
    OMX_VIDEO_HEVCProfileMainStillPicture  = 0x04,   /**< Main still picture
profile */
    OMX_VIDEO_HEVCProfileKhronosExtensions  =  0x6F000000,  /**<  Reserved
region for introducing Khronos Standard Extensions */
    OMX_VIDEO_HEVCProfileVendorStartUnused  =  0x7F000000,  /**<  Reserved
region for introducing Vendor Extensions */
    OMX_VIDEO_HEVCProfileMax      = 0x7FFFFFFF
} OMX_VIDEO_HEVCPROFILETYPE;
```

1.2.5 OMX_VIDEO_HEVCLEVELTYPE

Defines HEVC/H.265 video level types

```
typedef enum OMX_VIDEO_HEVCLEVELTYPE {
    OMX_VIDEO_HEVCLevel1   = 0x01,     /**< Level 1   */
    OMX_VIDEO_HEVCLevel2   = 0x02,     /**< Level 2   */
    OMX_VIDEO_HEVCLevel21  = 0x04,     /**< Level 2.1 */
    OMX_VIDEO_HEVCLevel3   = 0x08,     /**< Level 3   */
    OMX_VIDEO_HEVCLevel31  = 0x10,     /**< Level 3.1 */
    OMX_VIDEO_HEVCLevel4   = 0x20,     /**< Level 4   */
    OMX_VIDEO_HEVCLevel41  = 0x40,     /**< Level 4.1 */
    OMX_VIDEO_HEVCLevel5   = 0x80,     /**< Level 5   */
    OMX_VIDEO_HEVCLevel51  = 0x100,    /**< Level 5.1 */
    OMX_VIDEO_HEVCLevel52  = 0x200,    /**< Level 5.2 */
    OMX_VIDEO_HEVCLevel6   = 0x400,    /**< Level 6   */
    OMX_VIDEO_HEVCLevel61  = 0x800,    /**< Level 6.1 */
    OMX_VIDEO_HEVCLevel62  = 0x1000,   /**< Level 6.2 */
    OMX_VIDEO_HEVCLevelKhronosExtensions = 0x6F000000, /**< Reserved region
for introducing Khronos Standard Extensions */
    OMX_VIDEO_HEVCLevelVendorStartUnused = 0x7F000000, /**< Reserved region
for introducing Vendor Extensions */
```

```
    OMX_VIDEO_HEVCLevelMax = 0x7FFFFFFF
} OMX_VIDEO_HEVCLEVELTYPE;
```

1.2.6 OMX_VIDEO_VP9PROFILETYPE

Defines VP9 video profile types

```
typedef enum OMX_VIDEO_VP9PROFILETYPE {
    OMX_VIDEO_VP9Profile0 = 0x01, /* 8-bit 4:2:0 */
    OMX_VIDEO_VP9Profile1 = 0x02, /* 8-bit 4:2:2, 4:4:4, alpha channel */
    OMX_VIDEO_VP9Profile2 = 0x04, /* 10-bit/12-bit 4:2:0, YouTube Premium
Content Profile */
    OMX_VIDEO_VP9Profile3 = 0x08, /* 10-bit/12-bit 4:2:2, 4:4:4, alpha
channel */
    OMX_VIDEO_VP9ProfileUnknown = 0x6EFFFFFF,
    OMX_VIDEO_VP9ProfileKhronosExtensions = 0x6F000000, /**< Reserved region
for introducing Khronos Standard Extensions */
    OMX_VIDEO_VP9ProfileVendorStartUnused = 0x7F000000, /**< Reserved region
for introducing Vendor Extensions */
    OMX_VIDEO_VP9ProfileMax = 0x7FFFFFFF
} OMX_VIDEO_VP9PROFILETYPE;
```

1.2.7 OMX_VIDEO_G2PIXELFORMAT

Defines G2 decoder pixel formats

```
/** G2 Decoder pixel formats */
typedef enum OMX_VIDEO_G2PIXELFORMAT {
    OMX_VIDEO_G2PixelFormat_Default     = 0x0,
    OMX_VIDEO_G2PixelFormat_8bit        = 0x01, /* 10 bit data is clamped to
8 bit per pixel */
    OMX_VIDEO_G2PixelFormat_P010        = 0x02, /* MS P010 format */
    OMX_VIDEO_G2PixelFormat_Custom1     = 0x03
} OMX_VIDEO_G2PIXELFORMAT;
```

1.3 Data Structures

1.3.1 OMX_VIDEO_PARAM_MVCSTREAMTYPE

Structure for configuring G1 H.264 decoder to MVC mode

```
typedef struct OMX_VIDEO_PARAM_MVCSTREAMTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bIsMVCStream;
} OMX_VIDEO_PARAM_MVCSTREAMTYPE;
```

1.3.2 OMX_VIDEO_CONFIG_INTRAAREATYPE

Structure for configuring Intra area for 8290/H1/H2 encoder

```
typedef struct OMX_VIDEO_CONFIG_INTRAAREATYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
```

```
    OMX_U32 nPortIndex;
    OMX_BOOL bEnable;
    OMX_U32 nTop;       /* Top mb row inside area [0..heightMbs-1]      */
    OMX_U32 nLeft;      /* Left mb row inside area [0..widthMbs-1]      */
    OMX_U32 nBottom;    /* Bottom mb row inside area [top..heightMbs-1] */
    OMX_U32 nRight;     /* Right mb row inside area [left..widthMbs-1]  */
} OMX_VIDEO_CONFIG_INTRAAREATYPE;
```

1.3.3 OMX_VIDEO_CONFIG_ROIAREATYPE

Structure for configuring ROI area for 8290/H1/H2 encoder

```
typedef struct OMX_VIDEO_CONFIG_ROIAREATYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnable;
    OMX_U32 nArea;      /* ROI area number [1..2]                       */
    OMX_U32 nTop;       /* Top mb row inside area [0..heightMbs-1]      */
    OMX_U32 nLeft;      /* Left mb row inside area [0..widthMbs-1]      */
    OMX_U32 nBottom;    /* Bottom mb row inside area [top..heightMbs-1] */
    OMX_U32 nRight;     /* Right mb row inside area [left..widthMbs-1]  */
} OMX_VIDEO_CONFIG_ROIAREATYPE;
```

1.3.4 OMX_VIDEO_CONFIG_ROIDELTAQPTYPE

Structure for configuring ROI Delta QP for 8290/H1/H2 encoder

```
typedef struct OMX_VIDEO_CONFIG_ROIDELTAQPTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nArea;      /* ROI area number [1..2]               */
    OMX_S32 nDeltaQP;   /* QP delta value [-127..0] for VP8     */
                        /*                [-15..0]  for H264    */
} OMX_VIDEO_CONFIG_ROIDELTAQPTYPE;
```

1.3.5 OMX_VIDEO_CONFIG_ADAPTIVEROITYPE

Structure for configuring Adaptive ROI for H1 encoder

```
typedef struct OMX_VIDEO_CONFIG_ADAPTIVEROITYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_S32 nAdaptiveROI;     /* QP delta for adaptive ROI [-51..0]       */
    OMX_S32  nAdaptiveROIColor;   /* Color temperature for the Adaptive ROI
*/
                                /*-10=2000K,        0=3000K,        10=5000K
*/
} OMX_VIDEO_CONFIG_ADAPTIVEROITYPE;
```

1.3.6 OMX_VIDEO_CONFIG_VP8TEMPORALLAYERTYPE

Structure for configuring VP8 temporal layers

```
typedef struct OMX_VIDEO_CONFIG_VP8TEMPORALLAYERTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_U32 nBaseLayerBitrate;  /* Bits per second [10000..40000000]    */
    OMX_U32 nLayer1Bitrate;     /* Bits per second [10000..40000000]    */
    OMX_U32 nLayer2Bitrate;     /* Bits per second [10000..40000000]    */
    OMX_U32 nLayer3Bitrate;     /* Bits per second [10000..40000000]    */
} OMX_VIDEO_CONFIG_VP8TEMPORALLAYERTYPE;
```

1.3.7 OMX_VIDEO_PARAM_HEVCTYPE

Defines parameters for HEVC/H.265 video standard

```
typedef struct OMX_VIDEO_PARAM_HEVCTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_VIDEO_HEVCPROFILETYPE eProfile;
    OMX_VIDEO_HEVCLEVELTYPE eLevel;
    OMX_U32 nPFrames;
    OMX_U32 nRefFrames;
    OMX_U32 nBitDepthLuma;
    OMX_U32 nBitDepthChroma;
    OMX_BOOL bStrongIntraSmoothing;
    OMX_S32 nTcOffset;
    OMX_S32 nBetaOffset;
    OMX_BOOL bEnableDeblockOverride;
    OMX_BOOL bDeblockOverride;
    OMX_BOOL bEnableSAO;
    OMX_BOOL bEnableScalingList;
    OMX_BOOL bCabacInitFlag;
} OMX_VIDEO_PARAM_HEVCTYPE;
```

1.3.8 OMX_VIDEO_PARAM_VP9TYPE

Defines parameters for VP9 video standard

```
typedef struct OMX_VIDEO_PARAM_VP9TYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_VIDEO_VP9PROFILETYPE eProfile;
    OMX_U32 nBitDepthLuma;
    OMX_U32 nBitDepthChroma;
} OMX_VIDEO_PARAM_VP9TYPE;
```

1.3.9 OMX_VIDEO_PARAM_G2CONFIGTYPE

Structure for configuring G2 decoder

```
typedef struct OMX_VIDEO_PARAM_G2CONFIGTYPE {
    OMX_U32 nSize;
    OMX_VERSIONTYPE nVersion;
    OMX_U32 nPortIndex;
    OMX_BOOL bEnableTiled;
    OMX_VIDEO_G2PIXELFORMAT ePixelFormat;
```

```
    OMX_BOOL bEnableRFC;
} OMX_VIDEO_PARAM_G2CONFIGTYPE;
```

1.3.10 ALLOC_PRIVATE

This structure is used to communicate input/output buffer's physical address when buffers are allocated and when using FillThisBuffer or EmptyThisBuffer. The data is accessed through pInputPortPrivate pointer.

```
typedef struct ALLOC_PRIVATE {
    OMX_U8* pBufferData;                // Virtual address of the buffer
    OMX_U64 nBusAddress;                // Physical address of the buffer
    OMX_U32 nBufferSize;                // Allocated size
} ALLOC_PRIVATE;
```

1.3.11 RFC_TABLE

Structure for accessing Reference Frame Compression (RFC) table data

```
typedef struct RFC_TABLE {
    OMX_U8* pLumaBase;
    OMX_U64 nLumaBusAddress;
    OMX_U8* pChromaBase;
    OMX_U64 nChromaBusAddress;
} RFC_TABLE;
```

1.3.12 OUTPUT_BUFFER_PRIVATE

This structure is used for accessing output buffer's physical address and RFC table when buffer is returned to the client with FillBufferDone callback. The data is accessed through pOutputPortPrivate pointer. Separate luminance and chrominance buffers are required when VP9 decoder is used in tiled mode.

```
typedef struct OUTPUT_BUFFER_PRIVATE {
    OMX_U8* pLumaBase;          // Virtual address of the luminance buffer
    OMX_U64 nLumaBusAddress;    // Physical address of the luminance buffer
    OMX_U32 nLumaSize;          // Size of the luminance data
    OMX_U8* pChromaBase;        // Virtual address of the chrominance buffer
    OMX_U64 nChromaBusAddress; // Physical address of the chrominance buffer
    OMX_U32 nChromaSize;        // Size of the chrominance data
    RFC_TABLE sRfcTable;        // RFC table data
} OUTPUT_BUFFER_PRIVATE;
```

www.verisilicon.com