Qualcomm Technologies, Inc.

DragonBoard™ 410c based on Qualcomm® Snapdragon™ 410 processor

OMX Video Encoder - Android

June 2015

© 2015 Qualcomm Technologies, Inc. All rights reserved.

Qualcomm Snapdragon is a product of Qualcomm Technologies, Inc. Other Qualcomm products referenced herein are products of Qualcomm Technologies, Inc. or its other subsidiaries.

DragonBoard, Qualcomm and Snapdragon are trademarks of Qualcomm Incorporated, registered in the United States and other countries. All Qualcomm Incorporated trademarks are used with permission. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

Use of this document is subject to the license set forth in Exhibit 1.

Questions or comments: https://www.96boards.org/DragonBoard410c/forum

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

LM80-P0436-15 Rev B

Revision history

Revision	Date	Description
B	June 10, 2015	Revised OMX source code information & port definition
A	May 27, 2015	Initial release

Contents

Figures

Tables

1 Introduction

1.1 Purpose

This document is a reference for integrating hardware video decoders using the Qualcomm Technologies, Inc. implementation of the OpenMAX (OMX) Integration Layer (IL).

1.2 Scope

This document applies to devices using the APQ8016 processor.

It is intended for third-party implementers who have multimedia frameworks that use QTI hardware codecs for the video encoder.

1.3 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `#include`.

Code variables appear in angle brackets, for example, `<number>`.

Commands to be entered appear in a different font, e.g., **`copy a:*.* b:`**.

Shading indicates content that has been added or changed in this revision of the document.

1.4 Acronyms and terms

Acronym or term	Definition
IL	Integration Layer
OMX	OpenMAX
QTI	Qualcomm Technologies, Inc.

1.5 Additional information

For additional information, go to https://www.96boards.org/DragonBoard410c/docs.

2 OMX Components

2.1 OMX core

The OMX core component is the top-level interface exposed to the multimedia framework for use with all QTI hardware codecs.

This component supports the standard OMX IL interface specification for easy plug-in and is constructed as a dynamic library.

2.2 Integration procedure

The OMX core component is the only shared library that must be linked with the client's framework to use the QTI hardware video decoders.

This library is available in the Android build as `libOmxCore.so`.

2.3 OMX components for the video encoder

The OMX video encoder components are specific and are loaded by the OMX core at runtime based on a client request. These components are derived from a common interface and implement the functionality of the specific encoder.

3 OMX Sequence

3.1 Prerequisite

Before proceeding with the OMX sequence, link to the OMX core library listed in Section 2.2.

3.2 Determining support for the role

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

Figure 3-1 shows the flow for an IL client to query the names of all the installed components that support a given role.

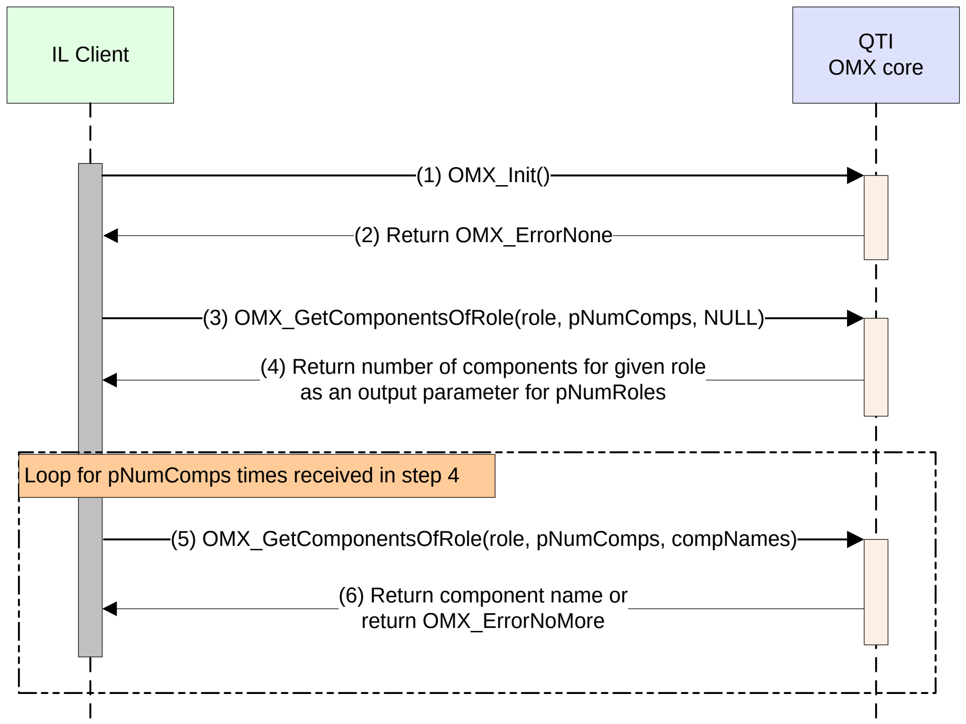

IL Client
QTI OMX core
(1) OMX_Init()
(2) Return OMX_ErrorNone
(3) OMX_GetComponentsOfRole(role, pNumComps, NULL)
(4) Return number of components for given role as an output parameter for pNumRoles
Loop for pNumComps times received in step 4
(5) OMX_GetComponentsOfRole(role, pNumComps, compNames)
(6)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

Figure 3-1 Sequence diagram for initialization

3.3 Loading the component

Figure 3-2 shows the `OMX_GetHandle` API used to get the component handle for the required role.

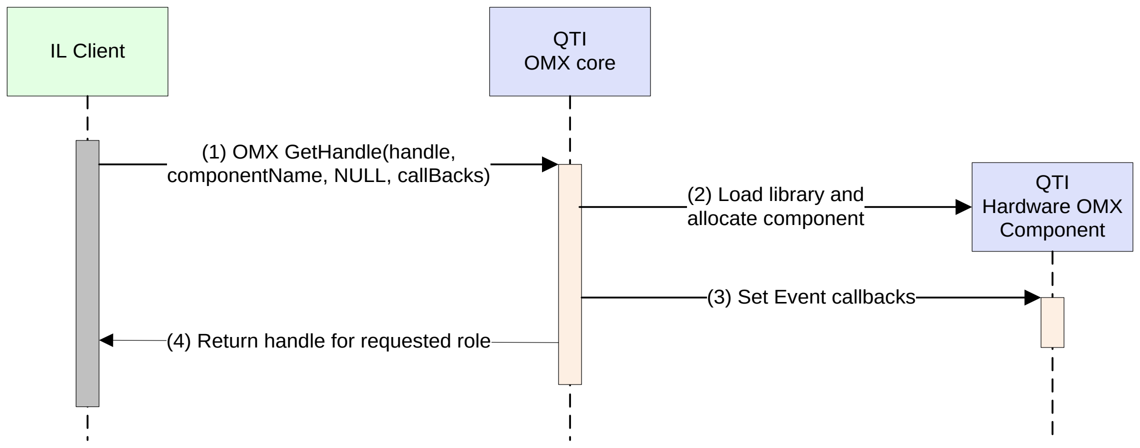

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX GetHandle(handle, componentName, NULL, callBacks)
(2) Load library and allocate component
(3) Set Event callbacks
(4) Return handle for requested role

Figure 3-2 Sequence diagram for loading a component

The numbers in parentheses in Figure 3-2 refer to the following steps:

1. The `OMX_GetHandle` call (1) loads the component library dynamically.
2. The call initializes the components (2).
3. Callback events (`OMX_CALLBACKTYPE`) are set to notify the client directly for `FillBufferDone`, `EmptyBufferDone`, and Events from the components (3).

3.4 Handshake configuration of the encoder component

After the component handle is acquired, the client initializes and configures the component by retrieving the port definition (OMX_PARAM_PORTDEFINITIONTYPE) and format of the components and setting them accordingly.

Clients are encouraged to use the component port definitions as much as possible because OMX components and video drivers are optimized for specific chipsets.

Figure 3-3 shows a minimum handshake between the IL client and the component for port configuration.

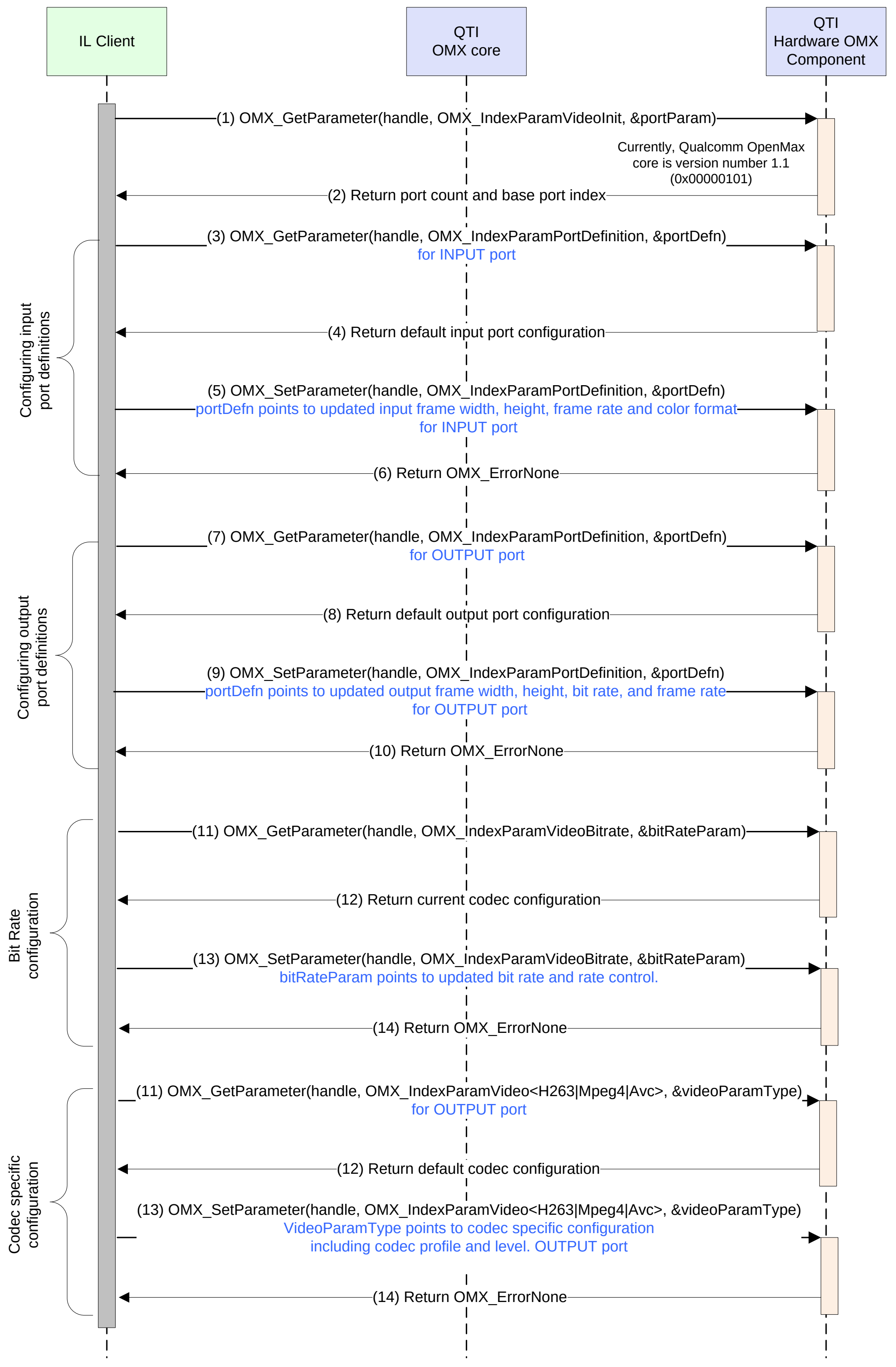

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_GetParameter(handle, OMX_IndexParamVideoInit, &portParam)
Currently, Qualcomm OpenMax core is version number 1.1 (0x00000101)
(2) Return port count and base port index
Configuring input port definitions
(3) OMX_GetParameter(handle, OMX_IndexParamPortDefinition, &portDefn)
for INPUT port
(4) Return default input port configuration
(5) OMX_SetParameter(handle, OMX_IndexParamPortDefinition, &portDefn)
portDefn points to updated input frame width, height, frame rate and color format
for INPUT port
(6) Return OMX_ErrorNone
Configuring output port definitions
(7) OMX_GetParameter(handle, OMX_IndexParamPortDefinition, &portDefn)
for OUTPUT port
(8) Return default output port configuration
(9) OMX_SetParameter(handle, OMX_IndexParamPortDefinition, &portDefn)
portDefn points to updated output frame width, height, bit rate, and frame rate
for OUTPUT port
(10) Return OMX_ErrorNone
Bit Rate configuration
(11) OMX_GetParameter(handle, OMX_IndexParamVideoBitrate, &bitRateParam)
(12) Return current codec configuration
(13) OMX_SetParameter(handle, OMX_IndexParamVideoBitrate, &bitRateParam)
bitRateParam points to updated bit rate and rate control.
(14) Return OMX_ErrorNone
Codec specific configuration
(11) OMX_GetParameter(handle, OMX_IndexParamVideo<H263|Mpeg4|Avc>, &videoParamType)
for OUTPUT port
(12) Return default codec configuration
(13) OMX_SetParameter(handle, OMX_IndexParamVideo<H263|Mpeg4|Avc>, &videoParamType)
VideoParamType points to codec specific configuration including codec profile and level. OUTPUT port
(14) Return OMX_ErrorNone

Figure 3-3 Sequence diagram for configuring a component port

The numbers in parentheses in Figure 3-3 refer to the following steps:

1. `OMX_GetParameter` (with `OMX_IndexParamVideoInit`) is called (1) to retrieve the number of ports supported by the QTI OMX component.
2. The component returns the number of ports supported to retrieve the individual port configuration (2).
3. For each port, the client can get the default configuration of the input and output ports of the component (3 and 7).
4. The IL client must call OMX_SetParameter to configure input nFrameWidth, nFrameHeight, eColorFormat, nBufferCountActual, xFramerate, and nBufferSize (5).

   The following table shows the configuration parameters that can be set using `OMX_IndexParamPortDefinition` for the input port.

Table 3-1 Parameters set with OMX_IndexParamPortDefinition (input port)

OMX_Index	OMX parameter	Comments
`OMX_IndexParam PortDefinition` (input port)	`OMX_PARAM_PORTDEFINITIONTYPE; .nBufferCountMin`	Minimum buffer count (obtained from `getparameter step)`
	`OMX_PARAM_PORTDEFINITIONTYPE; .nBufferCountActual`	Client should set this number to equal or greater than `nBufferCountMin` from the previous `OMX_GetParameter` call
	`OMX_VIDEO_PORTDEFINITIONTYPE; format.video.nFrameWidth`	Frame width
	`OMX_VIDEO_PORTDEFINITIONTYPE; format.video.nFrameHeight`	Frame height
	`OMX_VIDEO_PORTDEFINITIONTYPE; format.video.xFramerate`	Frame rate in Q16 format
	`OMX_VIDEO_PORTDEFINITIONTYPE; .nBufferSize`	Minimum buffer size in bytes allocated for this port (obtained from `getparameter` step)
	`OMX_VIDEO_PORTDEFINITIONTYPE; format.video.eColorFormat`	▪ OMX_COLOR_FormatYUV420 SemiPlanar→NV12 ▪ QOMX_COLOR_FormatYVU42 0SemiPlanar→NV21 ▪ QOMX_COLOR_Format AndroidOpaque→RGBA8888

5. The IL client must call the `OMX_SetParameter` to configure output `nFrameWidth`, `nFrameHeight`, `nBitrate`, and `xFramerate` (9).

The following table shows the configuration parameters that can be set using `OMX_IndexParamPortDefinition` for the output port.

Table 3-2 Parameters set with OMX_IndexParamPortDefinition (output port)

OMX_Index	OMX parameter	Comments
`OMX_IndexParam PortDefinition` (output port)	`OMX_PARAM_PORTDEFINITIONTYPE; .nBufferCountMin`	Minimum buffer count (obtained from `getparameter` step)
	`OMX_PARAM_PORTDEFINITIONTYPE; .nBufferCountActual`	Client should set this number to equal or greater than `nBufferCountMin` from the previous `OMX_GetParameter` call
	`OMX_VIDEO_PORTDEFINITIONTYPE; format.video.nFrameWidth`	Frame width
	`OMX_VIDEO_PORTDEFINITIONTYPE; format.video.nFrameHeight`	Frame height
	`OMX_VIDEO_PORTDEFINITIONTYPE; format.video.xFramerate`	Frame rate in Q16 format
	`OMX_VIDEO_PORTDEFINITIONTYPE; format.video.nBitrate`	Bitrate of the frame used on the port if data is compressed
	`OMX_VIDEO_PORTDEFINITIONTYPE; .nBufferSize`	Minimum buffer size in bytes allocated for this port (obtained from `getparameter` step)

6. The IL client must call `OMX_SetParameter` to configure the bitrate and type of rate control (13).

   The following table shows the configuration parameters that can be set using `OMX_IndexParamVideoBitrate`.

Table 3-3 Parameters set with OMX_IndexParamVideoBitrate

OMX_Index	OMX parameter	Comments
`OMX_IndexParam VideoBitrate` (output port)	`OMX_VIDEO_PARAM_BITRATETYPE; .eControlRate`	▪ OMX_Video_ControlRate Disable ▪ `OMX_Video_ControlRate Variable` (variable bitrate, constant frame rate) ▪ `OMX_Video_ControlRate ConstantSkipFrames` (constant bitrate, variable frame rate)
	`OMX_VIDEO_PARAM_BITRATETYPE; .nTargetBitrate`	Target bitrate for video encoding in bits per second

7. OMX_GetParameter might be called to get a default codec-specific configuration using index OMX_IndexParamVideoH263, OMX_IndexParamVideoMpeg4, or OMX_IndexParamVideoAvc.
8. The IL client calls `OMX_SetParameter` to configure the codec profile, level, and other codec-specific information into the OMX IL component (13). See Section 3.5 for other supported parameter configurations.

9. In the same manner, the codec profile and level can also be acquired and set by the IL Client with `OMX_IndexParamVideoProfileLevelCurrent`.

   The following table shows the configuration parameters that can be set using `OMX_IndexParamVideoProfileLevelCurrent`.

Table 3-4 Parameters set with OMX_IndexParamVideoProfileLevelCurrent

OMX_Index	OMX parameter	Comments
`OMX_IndexParam VideoProfileLevel Current` (output port)	`OMX_VIDEO_PARAM_PROFILELEVELTYPE; .eProfile`	Profile used
	`OMX_VIDEO_PARAM_PROFILELEVELTYPE; .eLevel`	Chosen processing level for a profile

3.5 Encoder configuration 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 bitrate, profile, and level. Figure 3-4 shows how to set specific parameters.

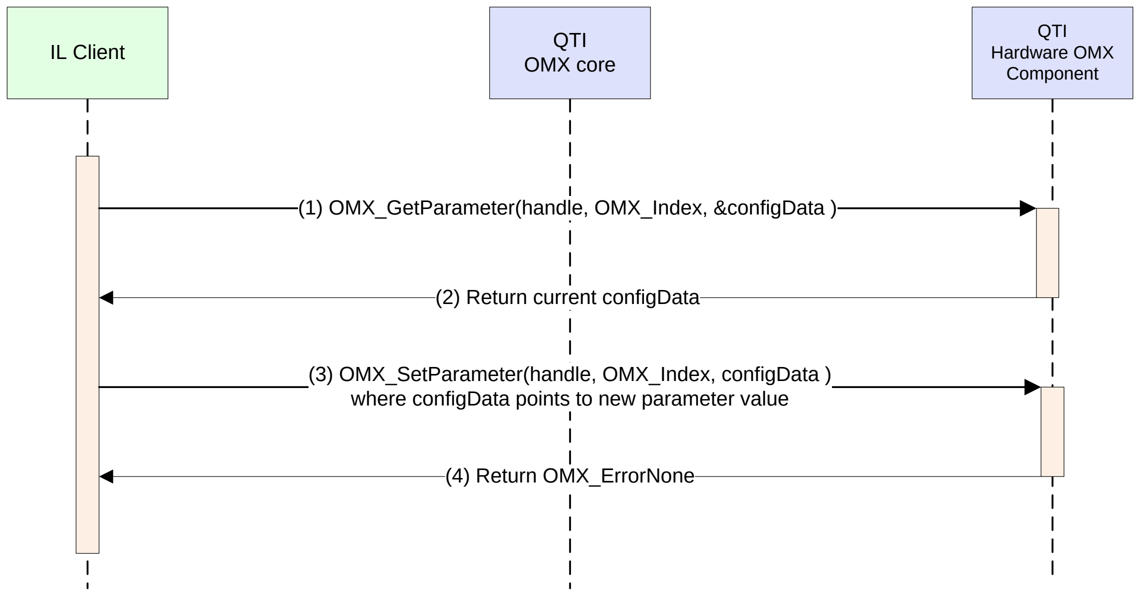

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_GetParameter(handle, OMX_Index, &configData)
(2) Return current configData
(3) OMX_SetParameter(handle, OMX_Index, configData)
where configData points to new parameter value
(4) Return OMX_ErrorNone

Figure 3-4 Sequence diagram for configuring encoder with OMX_Index parameter

The numbers in parentheses in Figure 3-4 refer to the following steps:

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

The table shows the configuration parameters that can be set using `OMX_Index`.

Table 3-5 Parameters set using OMX_Index

OMX_Index	OMX parameter	Comments
`OMX_IndexParamVideoPortFormat` (input port)	`OMX_VIDEO_PORTDEFINITIONTYPE; .eColorFormat`	▪ OMX_COLOR_FormatYUV420SemiPlanar→V4L2_PIX_FMT_NV12 ▪ QOMX_COLOR_FormatYVU420SemiPlanar→V4L2_PIX_FMT_NV21 ▪ QOMX_COLOR_Format AndroidOpaque→ RGBA8888
`OMX_IndexParamVideoPortFormat` (output port)	`OMX_VIDEO_PORTDEFINITIONTYPE; .xFramerate`	Frame rate in Q16 format
`OMX_IndexParamVideoErrorCorrection` (output port)	`OMX_VIDEO_PARAM_ERROR CORRECTIONTYPE; .bEnableHEC`	Enables header extension code
	`OMX_VIDEO_PARAM_ERROR CORRECTIONTYPE; .nResynchMarkerSpacing`	Resynchronization marker interval in bits
`OMX_IndexParamVideoIntraRefresh` (output port)	`OMX_VIDEO_PARAM_INTRAREFRESH TYPE; .eRefreshMode`	Cyclic intra-refresh→0
	`OMX_VIDEO_PARAM_INTRAREFRESH TYPE; .nCirMBs`	Number of consecutive macroblocks to be coded as intra when Cyclic intra-refresh is enabled (less than total number of MB in frame)
`OMX_QcomIndexParamVideoEncodeMetaBufferMode` (input port)	`StoreMetaDataInBuffersParams; .bStoreMetaData`	Enables usage of `MetaBufferMode` When metadata is enabled, information about the input buffer is passed instead of the actual contained data See Section 3.15 for more information

OMX_Index	OMX parameter	Comments
`OMX_QcomIndexParam IndexExtraDataType`	`QOMX_INDEXEXTRADATATYPE; .nIndex`	Possible values: `(OMX_INDEXTYPE)OMX_ ExtraDataVideoEncoder SliceInfo` `(OMX_INDEXTYPE)OMX_Extra DataVideoEncoderMBInfo` `(OMX_INDEXTYPE)OMX_Extra DataVideoLTRInfo`
	`QOMX_INDEXEXTRADATATYPE; .nPortIndex`	Port index information
`OMX_IndexParam StandardComponent Role`	`OMX_PARAM_COMPONENTROLE TYPE; .cRole`	Specifies codec
`OMX_QcomIndexParam VideoQPRange`	`OMX_QCOM_VIDEO_PARAM_ QPRANGETYPE; .minQP`	Sets the minimum quantization parameter used during encoding; recommended to leave default
	`OMX_QCOM_VIDEO_PARAM_ QPRANGETYPE; .maxQP`	Sets the maximum quantization parameter used during encoding; recommended to leave default
`OMX_IndexParam VideoQuantization`	`OMX_VIDEO_PARAM_ QUANTIZATIONTYPE; .nQpI`	Sets this quantization value for all I frames
	`OMX_VIDEO_PARAM_ QUANTIZATIONTYPE; .nQpP`	Sets this quantization value for all P frames
	`OMX_VIDEO_PARAM_ QUANTIZATIONTYPE; .nQpB`	Sets this quantization value for all B frames
`QOMX_IndexParam VideoInitialQp`	`OMX_EXTNINDEX_VIDEO_INIIALQP; .bEnableInitQP`	Enables this setting
	`OMX_EXTNINDEX_VIDEO_INIIALQP; .nQpI`	Sets this quantization value for just the first I frame
	`OMX_EXTNINDEX_VIDEO_INIIALQP; .nQpP`	Sets this quantization value for just the first P frame
	`OMX_EXTNINDEX_VIDEO_INIIALQP; .nQpB`	Sets this quantization value for just the first B frame
`OMX_QcomIndexParam SequenceHeaderWith IDR`	`PrependSPSPPSToIDRFrames Params; .bEnable`	Prepends SPS/PPS with every IDR frame
`OMX_QcomIndexParam H264AUDelimiter` (H.264 only)	`OMX_QCOM_VIDEO_CONFIG_H264_ AUD; .bEnable`	Adds a delimiter for separating metadata information from payload
`OMX_QcomIndex Hierarchical Structure` (H.264 and VP8 only)	`QOMX_VIDEO_HIERARCHICAL LAYERS; .eHierarchicalCodingType`	Possible values: ▪ QOMX_HIERARCHICAL CODING_P ▪ QOMX_HIERARCHICAL CODING_B
	`QOMX_VIDEO_HIERARCHICAL LAYERS; .nNumLayers`	Number of hierarchical layers count

OMX_Index	OMX parameter	Comments
`OMX_QcomIndexParamVideoLTRCount` (output port) (H.264 only)	`OMX_QCOM_VIDEO_PARAM_LTRCOUNT_TYPE; .nCount`	Enables LTR mode with up to .nCount LTR frames
`OMX_QcomIndexParamPerfLevel`	`OMX_QCOM_VIDEO_PARAM_PERF_LEVEL; ePerfLevel`	Possible values are: ▪ OMX_QCOM_PerfLevel Nominal ▪ `OMX_QCOM_PerfLevelTurbo`; only available on KitKat product lines
`OMX_QcomIndexParamH264VUITimingInfo`	`OMX_QCOM_VIDEO_PARAM_VUI_TIMING_INFO; .bEnable`	Enables and disables H.264 video usability information; only available on KitKat and later product lines
`OMX_QcomIndexParamPeakBitrate`	`OMX_QcomIndexParamPeakBitrate; .nPeakBitrate`	Limits the maximum bitrate of the encoded video to the value specified; only available on KitKat and later product lines

Figure 3-5 and Figure 3-6 show sample call flows for using the `OMX_SetParameter` call with various parameters from the above table.

Figure 3-5 shows the `OMX_SetParameter` call flow for configuring `eColorFormat` using `OMX_IndexParamVideoPortFormat`.

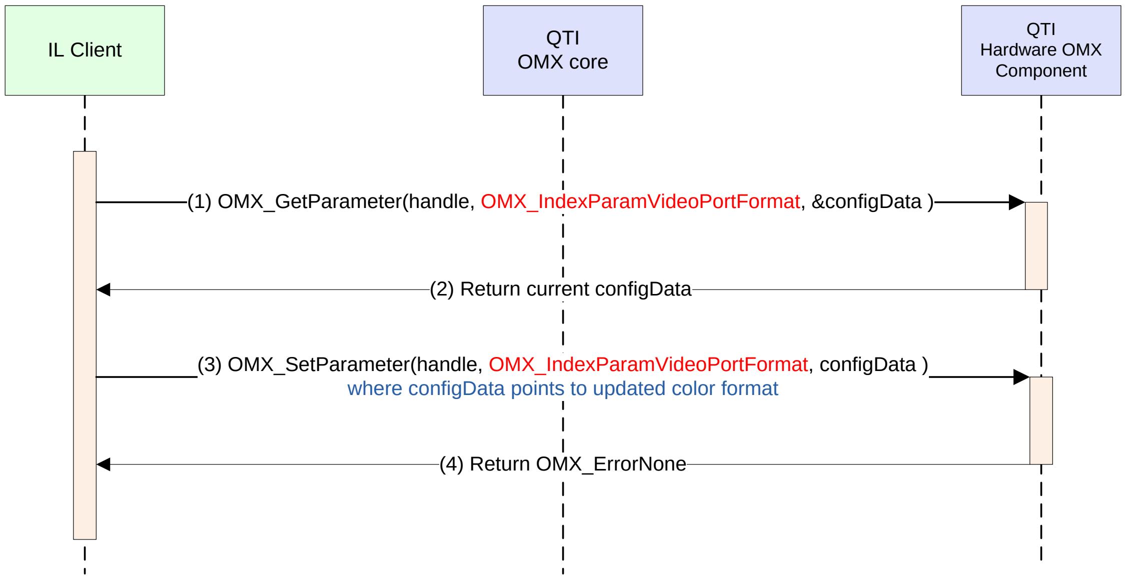

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_GetParameter(handle, OMX_IndexParamVideoPortFormat, &configData)
(2) Return current configData
(3) OMX_SetParameter(handle, OMX_IndexParamVideoPortFormat, configData)
where configData points to updated color format
(4) Return OMX_ErrorNone

Figure 3-5 OMX_SetParameter call flow for configuring eColorFormat

Figure 3-6 shows the `OMX_SetParameter` call flow for configuring `nQpI` and `nQpP` using `OMX_IndexParamVideoQuantization`.

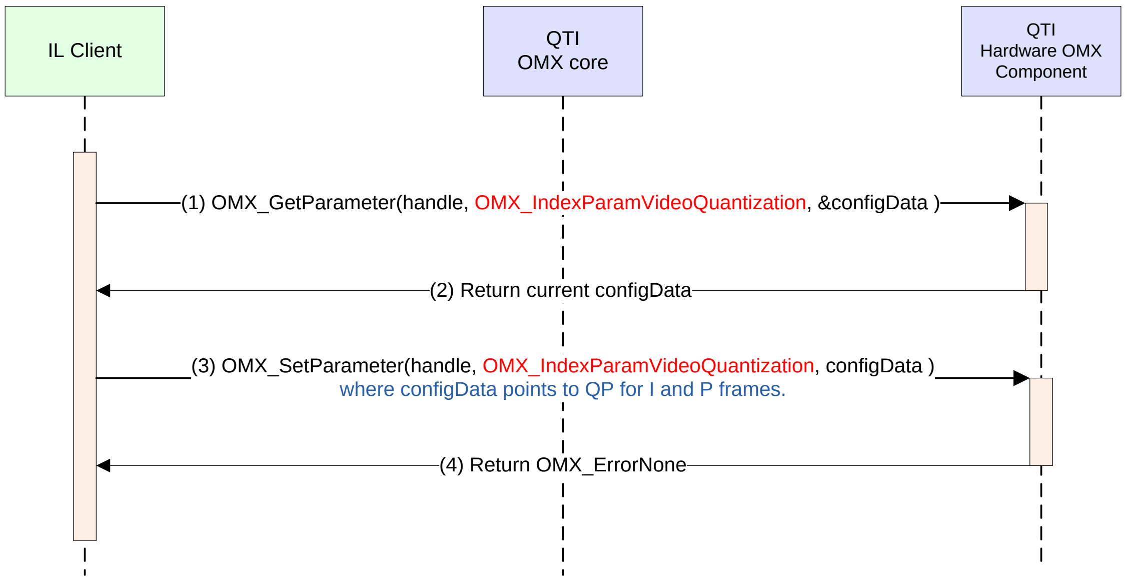

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_GetParameter(handle, OMX_IndexParamVideoQuantization, &configData)
(2) Return current configData
(3) OMX_SetParameter(handle, OMX_IndexParamVideoQuantization, configData)
where configData points to QP for I and P frames.
(4) Return OMX_ErrorNone

Figure 3-6 OMX_SetParameter call flow for configuring nQpI and nQpP

3.6 H.263 encoder-specific configuration parameters

QTI recommends using the H.263 encoder for the videotelephony application because it has a superior error recovery mechanism.

The encoder client must call the function `OMX_SetParameter` (see Figure 3-7) using `OMX_IndexParamVideoH263` to pass the specific H.263 configuration, as shown in Table 3-6.

Table 3-6 H.263 encoder-specific configuration parameters

Index	OMX parameter	Comments
`OMX_IndexParamVideoH263` (output port)	`OMX_VIDEO_PARAM_H263TYPE; .eProfile`	H263 profile
	`OMX_VIDEO_PARAM_H263TYPE; .eLevel`	Maximum processing level supported for a profile
	`OMX_VIDEO_PARAM_H263TYPE; .nPFrames`	Intraperiod; number of P frames within I frames

Figure 3-7 shows the `OMX_SetParameter` call flow for configuring `eProfile`, `eLevel`, and `nPFrames` using `OMX_IndexParamVideoH263`.

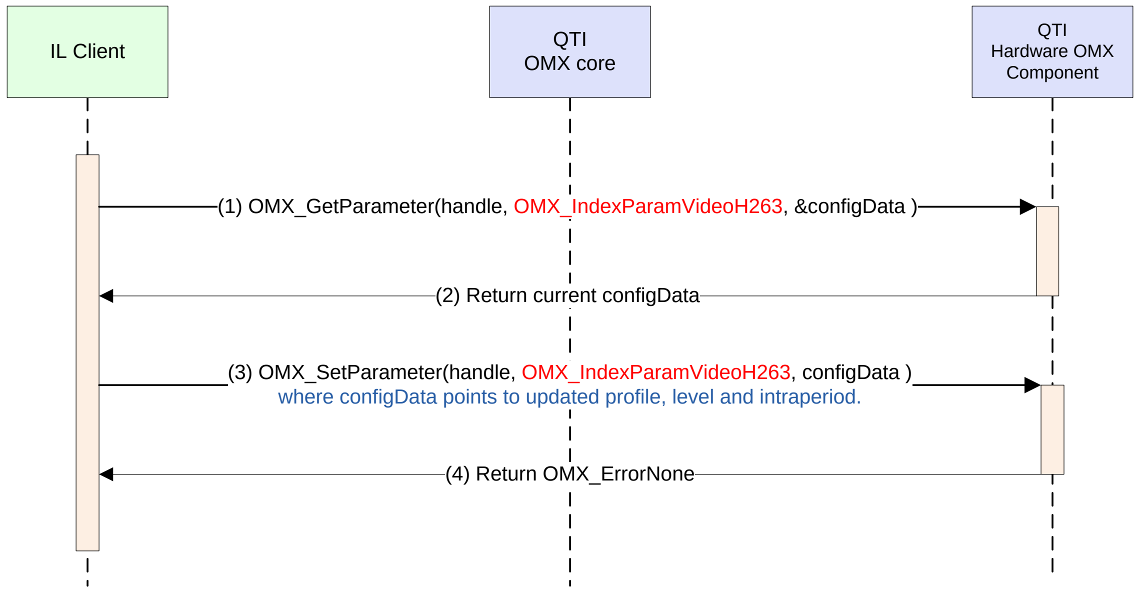

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_GetParameter(handle, OMX_IndexParamVideoH263, &configData)
(2) Return current configData
(3) OMX_SetParameter(handle, OMX_IndexParamVideoH263, configData)
where configData points to updated profile, level and intraperiod.
(4) Return OMX_ErrorNone

Figure 3-7 OMX_SetParameter call flow for configuring eProfile, eLevel, and nPFrames

3.7 MPEG-4-specific configuration parameter

The IL client can call `OMX_SetParameter` with the index `OMX_IndexParamVideoMpeg4` for setting any specific MPEG-4 configuration, as shown in Table 3-7.

Table 3-7 MPEG-4-specific configuration parameter

Index	OMX parameter	Comments
`OMX_IndexParamVideoMpeg4` (output port)	`OMX_VIDEO_PARAM_MPEG4TYPE;.eProfile`	MPEG-4 profile
	`OMX_VIDEO_PARAM_MPEG4TYPE;.eLevel`	Maximum processing level supported for a profile
	`OMX_VIDEO_PARAM_MPEG4TYPE;.nPFrames`	Intraperiod; number of P frames within I frames
	`OMX_VIDEO_PARAM_MPEG4TYPE;.nBFrames`	Intraperiod; number of B frames within I frames
	`OMX_VIDEO_PARAM_MPEG4TYPE;.nTimeIncRes`	VOP time increment resolution for MPEG-4
	`OMX_VIDEO_PARAM_MPEG4TYPE;.nSliceHeaderSpacing`	Number of macroblocks in a slice; make zero if not used

Figure 3-8 shows the `OMX_SetParameter` call flow for configuring `eProfile`, `eLevel`, `nPFrames`, `nTimeIncRes`, and `nSliceHeaderSpacing` using `OMX_IndexParamVideoMpeg4`.

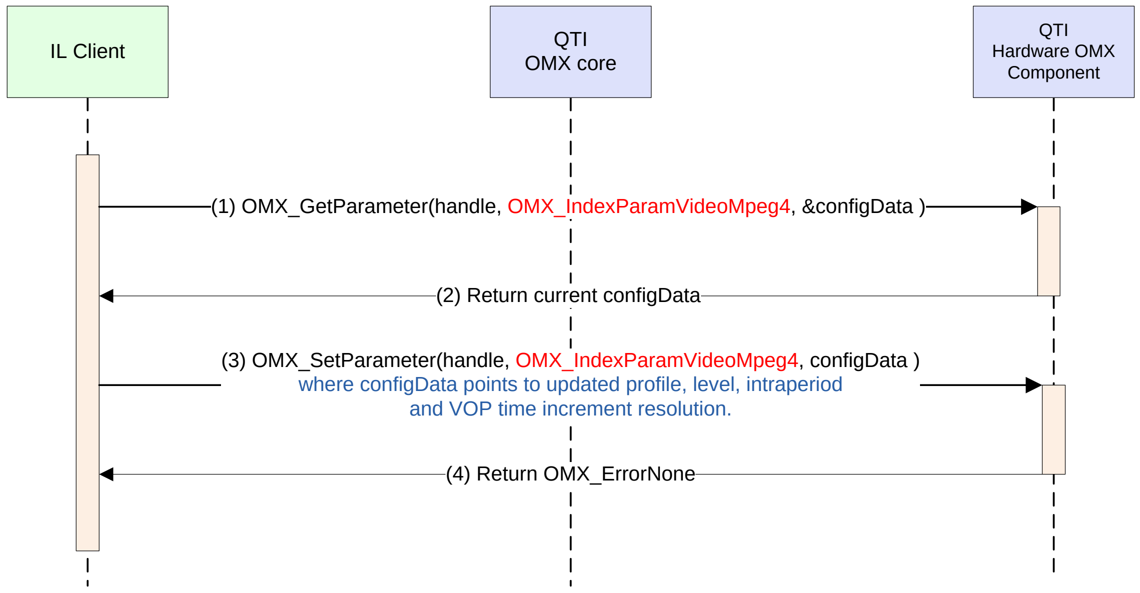

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_GetParameter(handle, OMX_IndexParamVideoMpeg4, &configData)
(2) Return current configData
(3) OMX_SetParameter(handle, OMX_IndexParamVideoMpeg4, configData)
where configData points to updated profile, level, intraperiod and VOP time increment resolution.
(4) Return OMX_ErrorNone

Figure 3-8 OMX_SetParameter call flow for configuring eProfile, eLevel, nPFrames, nTimeIncRes, and nSliceHeaderSpacing

3.8 VP8 encoder configuration

The encoder client might call the function `OMX_SetParameter` using `OMX_IndexParamVideoVp8` for setting the VP8 configuration, as shown in Table 3-8.

Table 3-8 VP8 encoder configuration

Index	OMX parameter	Comments
`OMX_IndexParamVideoVp8` (output port)	`OMX_VIDEO_PARAM_VP8TYPE; .eProfile`	VP8 profile
	`OMX_VIDEO_PARAM_VP8TYPE; .eLevel`	Maximum processing level supported for a profile

Figure 3-9 shows the `OMX_SetParameter` call flow for configuring `eProfile` and `eLevel` using `OMX_IndexParamVideoVp8`.

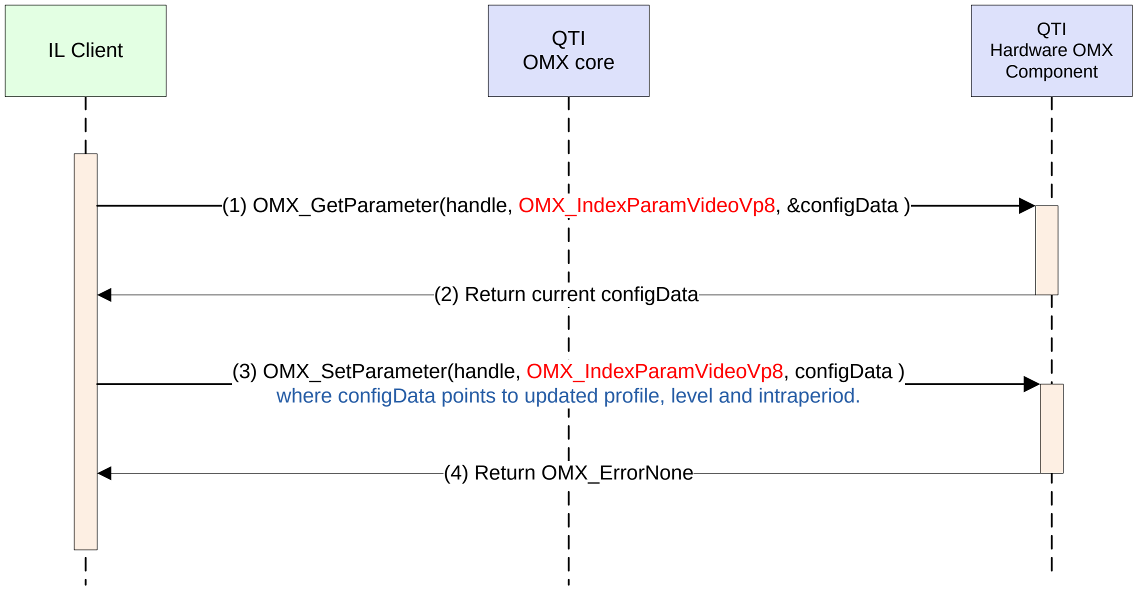

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_GetParameter(handle, OMX_IndexParamVideoVp8, &configData)
(2) Return current configData
(3) OMX_SetParameter(handle, OMX_IndexParamVideoVp8, configData)
where configData points to updated profile, level and intraperiod.
(4) Return OMX_ErrorNone

Figure 3-9 OMX_SetParameter call flow for configuring eProfile and eLevel

3.9 Buffer allocation

After configuring the component, the buffers are allocated. QTI hardware components use physical memory for buffers that interface with the codec accelerator. Figure 3-10 shows the buffer allocation sequence model.

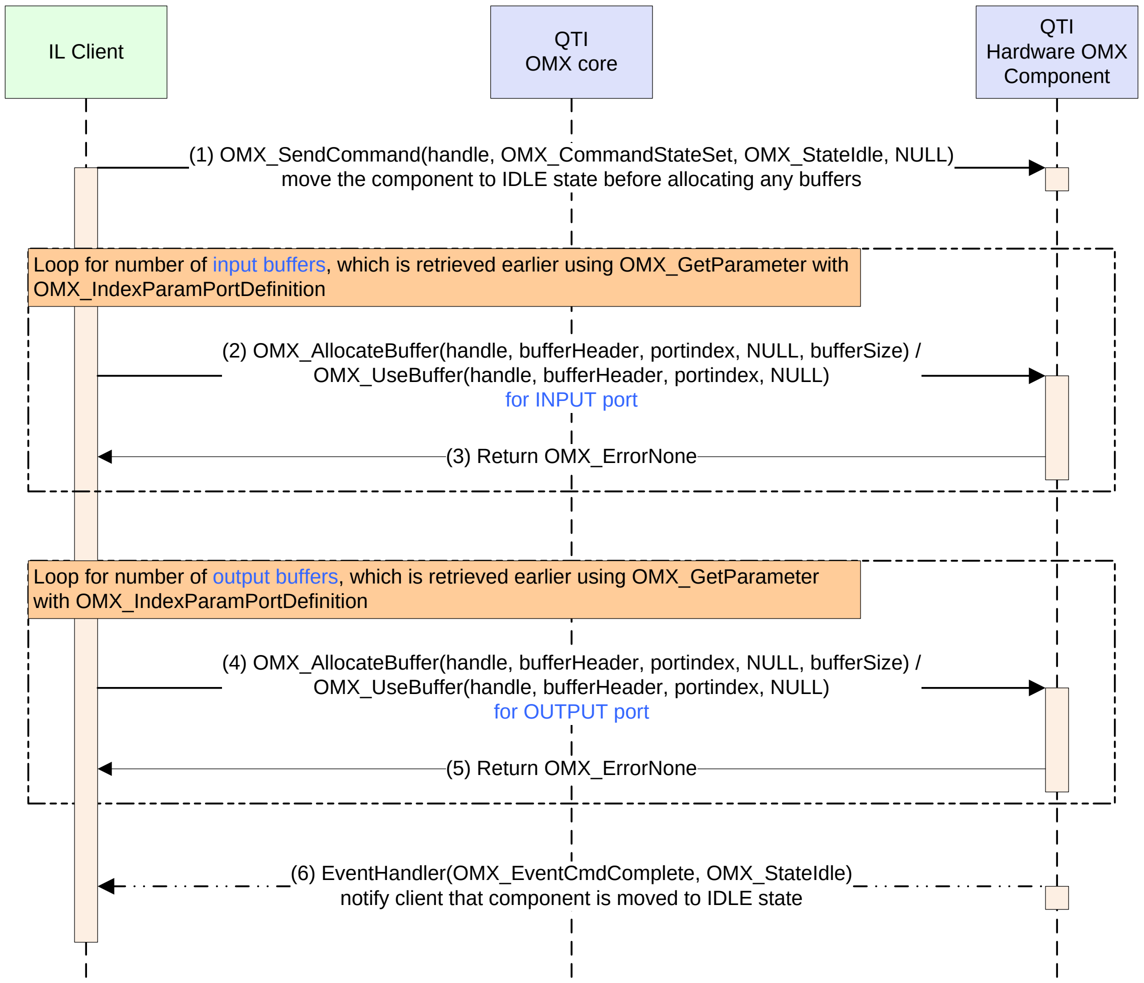

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_SendCommand(handle, OMX_CommandStateSet, OMX_StateIdle, NULL)
move the component to IDLE state before allocating any buffers
Loop for number of input buffers, which is retrieved earlier using OMX_GetParameter with OMX_IndexParamPortDefinition
(2) OMX_AllocateBuffer(handle, bufferHeader, portindex, NULL, bufferSize) / OMX_UseBuffer(handle, bufferHeader, portindex, NULL)
for INPUT port
(3) Return OMX_ErrorNone
Loop for number of output buffers, which is retrieved earlier using OMX_GetParameter with OMX_IndexParamPortDefinition
(4) OMX_AllocateBuffer(handle, bufferHeader, portindex, NULL, bufferSize) / OMX_UseBuffer(handle, bufferHeader, portindex, NULL)
for OUTPUT port
(5) Return OMX_ErrorNone
(6) EventHandler(OMX_EventCmdComplete, OMX_StateIdle)
notify client that component is moved to IDLE state

Figure 3-10 Sequence diagram for buffer allocation

The numbers in parentheses in Figure 3-10 refer to the following steps:

1. The components are moved from the Loaded state to the Idle state (1) to allocate the buffer.
2. On the input port, the client can use either `OMX_UseBuffer` or `OMX_AllocateBuffer` calls to the OMX component (2). This should be called for the number of input buffers in a loop.
3. QTI's camera solution can allocate physical contiguous memory and provide the color format required. For better performance, use `OMX_UseBuffer` on the input port, which avoids the memory copy of input YUV frames and saves memory usage and MIPS.
4. On the output port, the client can use either `OMX_UseBuffer` or `OMX_AllocateBuffer` calls to the OMX component (4). This should be called for the number of input buffers in a loop.

5. After 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` (6).

3.10 Data processing

After the component is ready for encoding 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.

The OMX component expects correct and updated timestamps in each input buffer, `OMX_BUFFERHEADERTYPE.nTimeStamp`, because they are used for target frame rate and I/P frame frequency calculations.

Figure 3-11 shows the sequence of calls for processing the input bitstream.

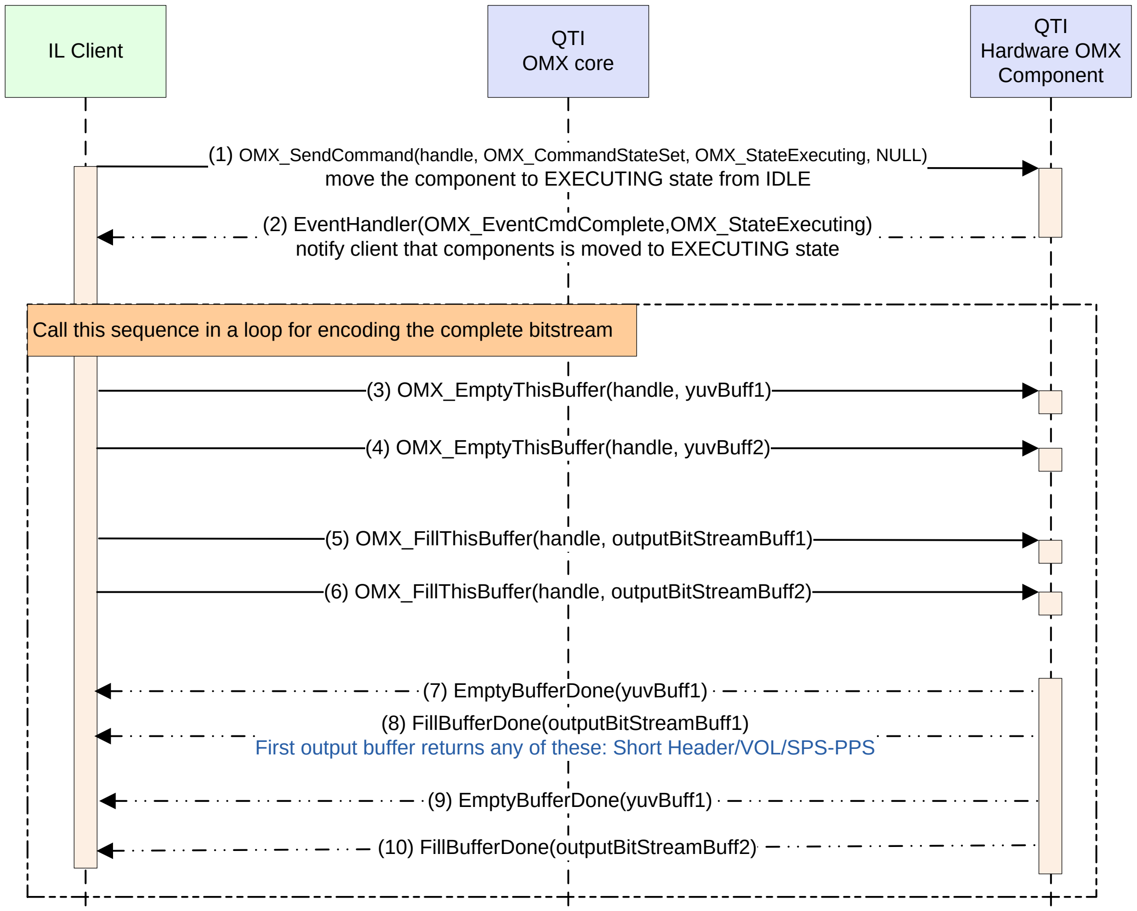

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_SendCommand(handle, OMX_CommandStateSet, OMX_StateExecuting, NULL) move the component to EXECUTING state from IDLE
(2) EventHandler(OMX_EventCmdComplete,OMX_StateExecuting) notify client that components is moved to EXECUTING state
Call this sequence in a loop for encoding the complete bitstream
(3) OMX_EmptyThisBuffer(handle, yuvBuff1)
(4) OMX_EmptyThisBuffer(handle, yuvBuff2)
(5) OMX_FillThisBuffer(handle, outputBitStreamBuff1)
(6) OMX_FillThisBuffer(handle, outputBitStreamBuff2)
(7) EmptyBufferDone(yuvBuff1)
(8) FillBufferDone(outputBitStreamBuff1)
First output buffer returns any of these: Short Header/VOL/SPS-PPS
(9) EmptyBufferDone(yuvBuff1)
(10) FillBufferDone(outputBitStreamBuff2)

Figure 3-11 Sequence diagram for data flow

The numbers in parentheses in Figure 3-11 refer to the following steps:

1. Transition the component from the Idle state to the Executing state (1).
2. Wait for OMX_EventCmdComplete (2).

3. After the IL client receives the command complete event for state transition, it can start sending data to the component.
4. Call `OMX_EmptyThisBuffer` (3 and 4) with the YUV data that is to be encoded.
5. Call `OMX_FillThisBuffer` (5 and 6) with the output buffers that can hold the video bitstream data.
6. The component generates the `EmptyBufferDone` (7 and 9) and `FillBufferDone` (8 and 10) callbacks to notify the client after processing the data.
7. The first `FillThisBufferDone` callback points to a buffer with the bitstream header information.
   - H263 – Short header + first frame
   - MPEG4 – VOL header
   - H264 – Sequence parameter set and picture parameter set
   - VP8 – Associated VP8 header

3.11 Dynamic configuration

Figure 3-12 shows changing parameters dynamically using `OMX_SetConfig`.

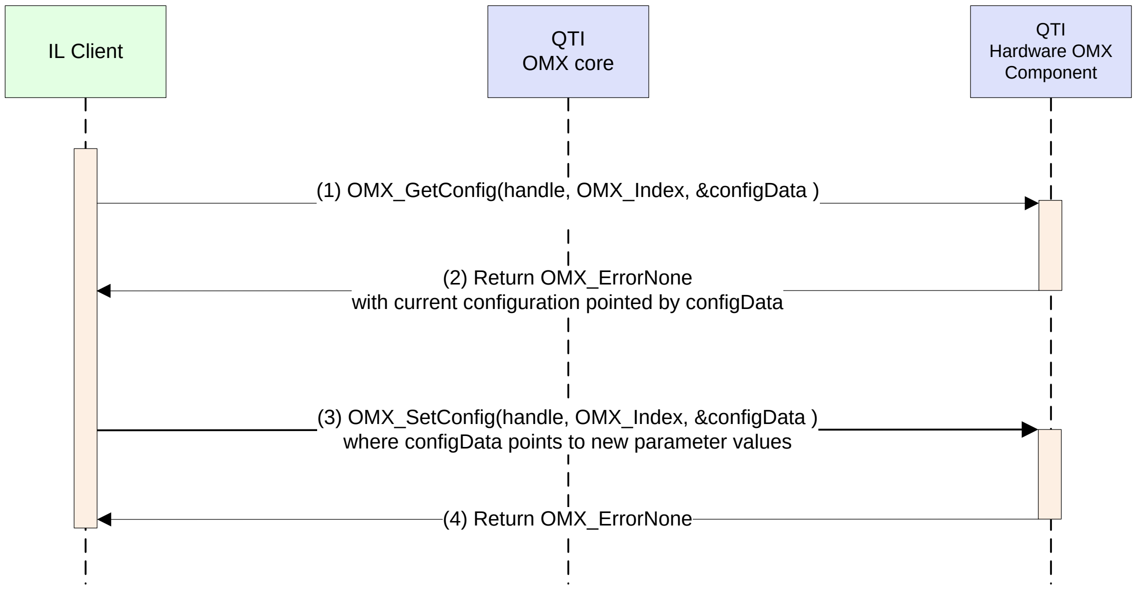

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_GetConfig(handle, OMX_Index, &configData)
(2) Return OMX_ErrorNone with current configuration pointed by configData
(3) OMX_SetConfig(handle, OMX_Index, &configData) where configData points to new parameter values
(4) Return OMX_ErrorNone

Figure 3-12 Sequence diagram for dynamically changing parameters

Table 3-9 shows the configuration parameters that can be set dynamically with `OMX_SetConfig`.

Table 3-9 Parameters set dynamically with OMX_SetConfig

Index	OMX parameter	Comments
`OMX_IndexConfigVideoBitrate` (output port)	`OMX_VIDEO_CONFIG_BITRATETYPE;` `.nEncodeBitrate`	Bitrate
`OMX_IndexConfigVideoIntraVOPRefresh` (output port)	`OMX_CONFIG_INTRAREFRESHVOPTYPE;` `.IntraRefreshVOP`	If `IntraRefreshVOP` = `OMX_TRUE;`, dynamically request future output buffer to be I frame
`QOMX_IndexConfigVideoIntraperiod` (output port)	`QOMX_VIDEO_INTRAPERIODTYPE;` `.nPFrames` `QOMX_VIDEO_INTRAPERIODTYPE;` `.nBFrames`	Dynamically change the intraperiod based on P and B frames (B frames when applicable)
`OMX_IndexConfigVideoFramerate` (output port)	`OMX_CONFIG_FRAMERATETYPE;` `.xEncodeFramerate`	Change the frame rate; encoder does not force output frame rate, but frame rate number is used for rate control calculations
`OMX_IndexConfigVideoAVCIntraPeriod`	`OMX_VIDEO_CONFIG_AVCINTRAPERIOD;` `.nPFrames` `OMX_VIDEO_CONFIG_AVCINTRAPERIOD;` `.nIDRPeriod`	Dynamically changes IDR period based on P and B frames (B frames when applicable) for AVC
`OMX_IndexConfigVideoVp8ReferenceFrame` (input port) (VP8 only)	`OMX_VIDEO_VP8REFERENCEFRAMETYPE;` `.nPortIndex` `.bUseGoldenFrame` `.bGoldenFrameRefresh`	Setting .bUseGoldenFrame will command the current frame to reference the golden frame instead of the previous reference frame. Setting .bGoldenFrameRefresh will use the current frame to replace the old Golden Frame as the new Golden frame.
`OMX_QcomIndexConfigVideoLTRUse` (input port) (H.264 only)	`OMX_QCOM_VIDEO_CONFIG_LTRUSE_TYPE;` `.nPortIndex` `.nID`	Will command the current frame to reference the LTR frame with ID (nID) instead of previous frame
`OMX_QcomIndexConfigVideoLTRMark` (input port) (H.264 only)	`OMX_QCOM_VIDEO_CONFIG_LTRMARK_TYPE;` `.nPortIndex` `.nID`	Will use current frame to replace the old LTR frame with ID (nID) as new LTR frame

Sample call flows for using the `OMX_SetConfiguration` call with various parameters from the above table are shown in Figure 3-13 through Figure 3-16.

Figure 3-13 shows the `OMX_SetConfiguration` call flow for setting `nEncodeBitrate` using `OMX_IndexConfigVideoBitrate`.

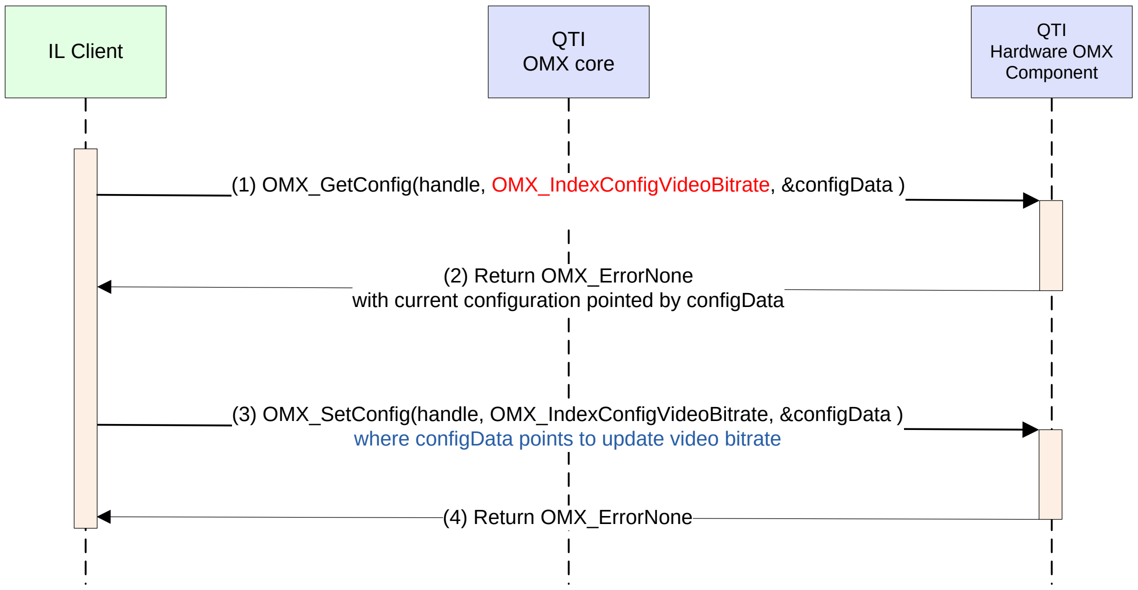

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_GetConfig(handle, OMX_IndexConfigVideoBitrate, &configData)
(2) Return OMX_ErrorNone with current configuration pointed by configData
(3) OMX_SetConfig(handle, OMX_IndexConfigVideoBitrate, &configData) where configData points to update video bitrate
(4) Return OMX_ErrorNone

Figure 3-13 OMX_SetConfiguration call flow for setting nEncodeBitrate

Figure 3-14 shows the `OMX_SetConfiguration` call flow for setting `IntraRefreshVOP` using `OMX_IndexConfigVideoIntraVOPRefresh`.

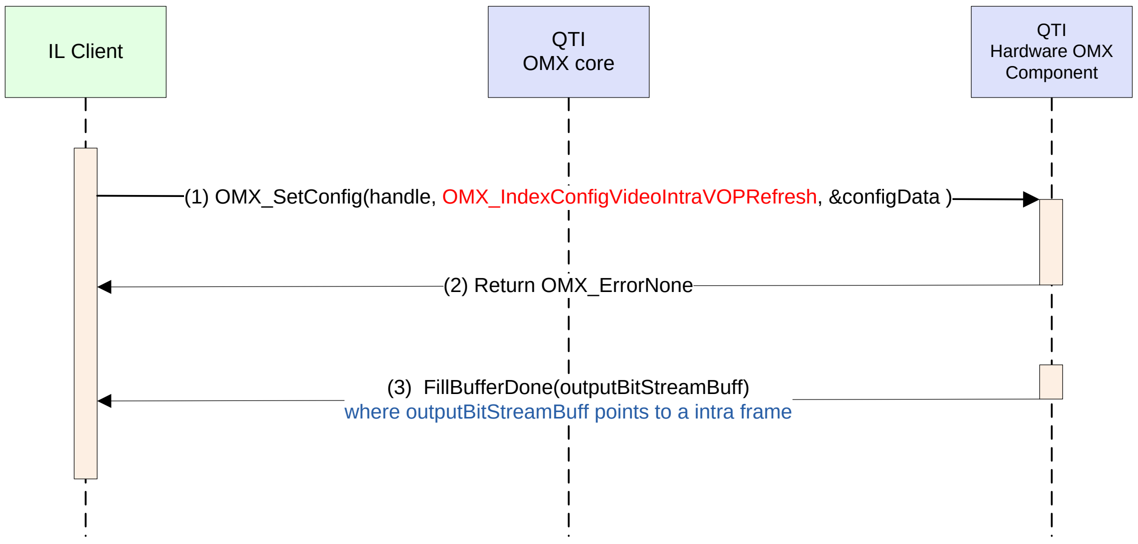

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_SetConfig(handle, OMX_IndexConfigVideoIntraVOPRefresh, &configData)
(2) Return OMX_ErrorNone
(3) FillBufferDone(outputBitStreamBuff) where outputBitStreamBuff points to a intra frame

Figure 3-14 OMX_SetConfiguration call flow for setting IntraRefreshVOP

Figure 3-15 shows the `OMX_SetConfiguration` call flow for setting `nPFrames` per frame using `QOMX_IndexConfigVideoIntraperiod`.

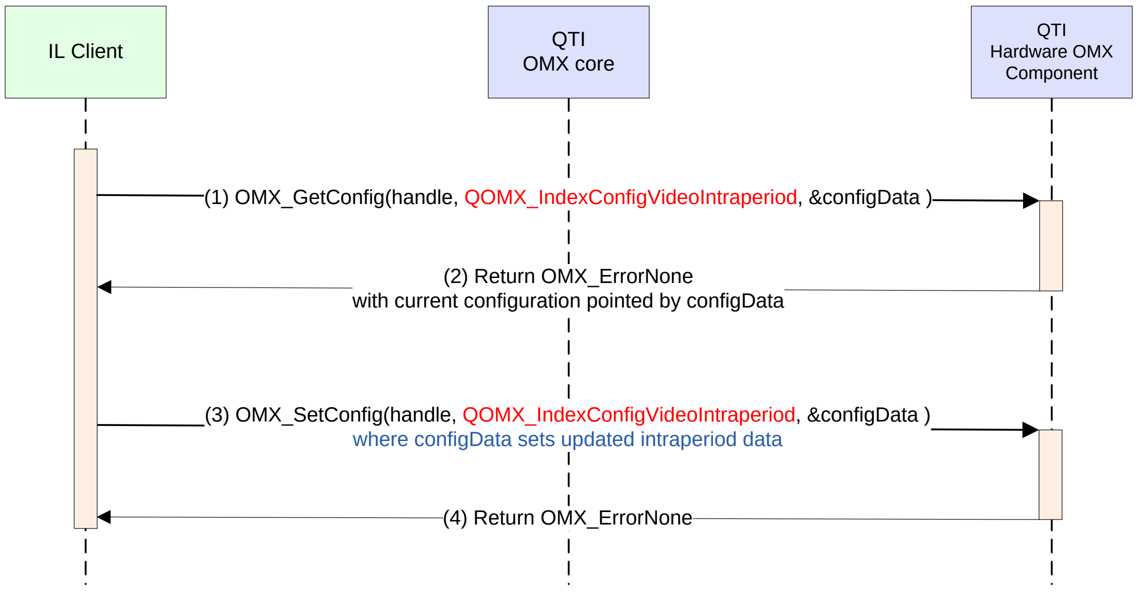

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_GetConfig(handle, QOMX_IndexConfigVideoIntraperiod, &configData)
(2) Return OMX_ErrorNone with current configuration pointed by configData
(3) OMX_SetConfig(handle, QOMX_IndexConfigVideoIntraperiod, &configData) where configData sets updated intraperiod data
(4) Return OMX_ErrorNone

Figure 3-15 OMX_SetConfiguration call flow for setting nPFrames per frame

Figure 3-16 shows the `OMX_SetConfiguration` call flow for setting `xEncodeFrameRate` using `OMX_IndexConfigVideoFramerate`.

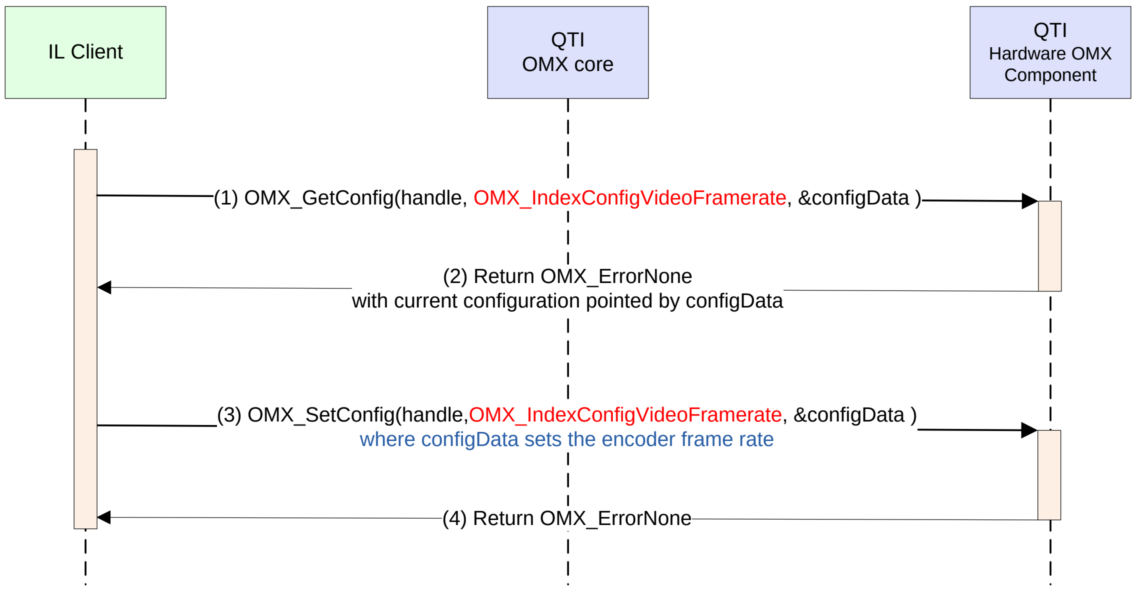

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_GetConfig(handle, OMX_IndexConfigVideoFramerate, &configData)
(2) Return OMX_ErrorNone with current configuration pointed by configData
(3) OMX_SetConfig(handle,OMX_IndexConfigVideoFramerate, &configData) where configData sets the encoder frame rate
(4) Return OMX_ErrorNone

Figure 3-16 OMX_SetConfiguration call flow for setting xEncodeFramerate

3.12 Deinitializing the component and OMX core

To deinitialize the OMX core, active components must be freed by moving them to the Execute→Idle→Loaded state, then freeing the input and output buffers. Figure 3-17 shows the teardown process of the component and OMX core.

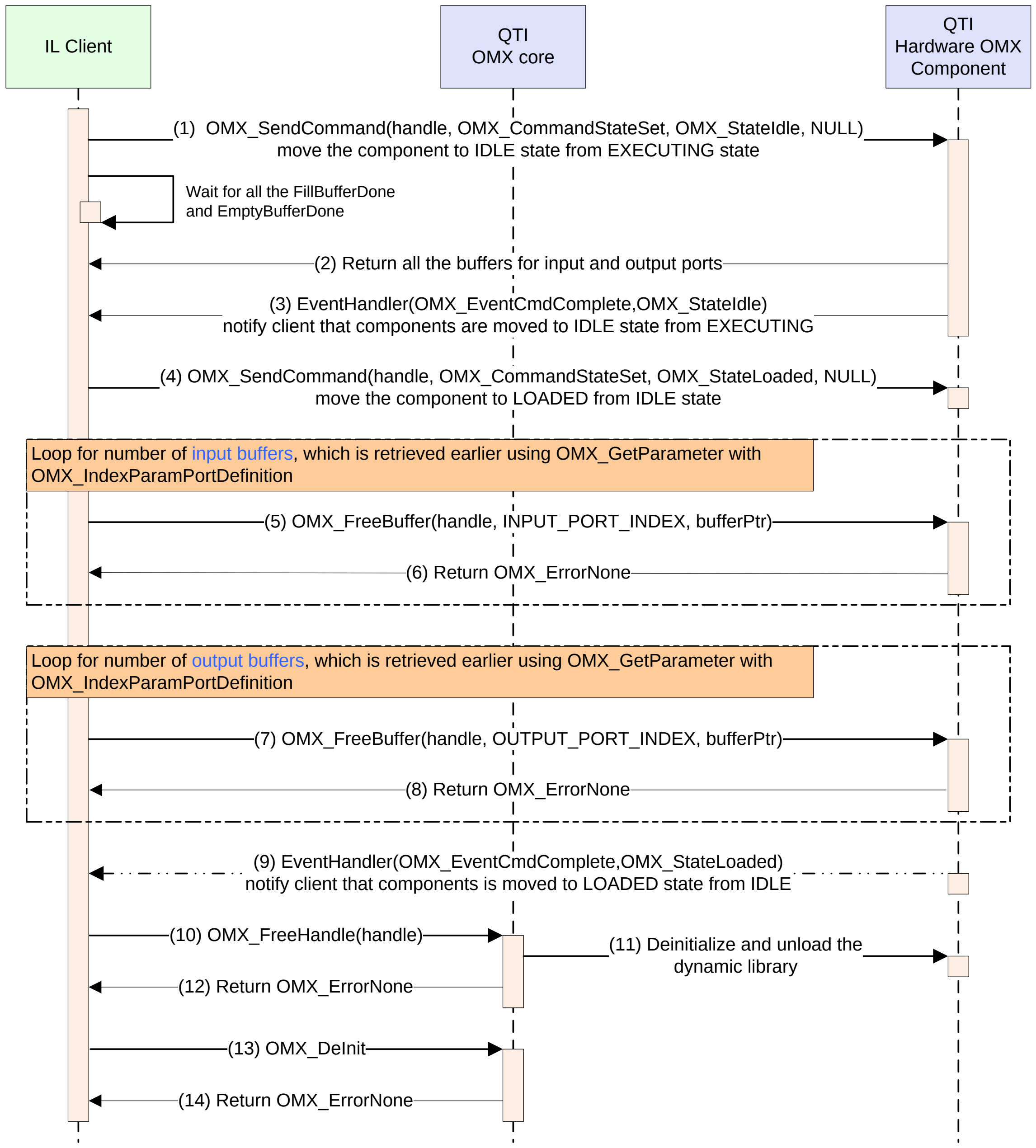

IL Client
QTI OMX core
QTI Hardware OMX Component
(1) OMX_SendCommand(handle, OMX_CommandStateSet, OMX_StateIdle, NULL) move the component to IDLE state from EXECUTING state
Wait for all the FillBufferDone and EmptyBufferDone
(2) Return all the buffers for input and output ports
(3) EventHandler(OMX_EventCmdComplete,OMX_StateIdle) notify client that components are moved to IDLE state from EXECUTING
(4) OMX_SendCommand(handle, OMX_CommandStateSet, OMX_StateLoaded, NULL) move the component to LOADED from IDLE state
Loop for number of input buffers, which is retrieved earlier using OMX_GetParameter with OMX_IndexParamPortDefinition
(5) OMX_FreeBuffer(handle, INPUT_PORT_INDEX, bufferPtr)
(6) Return OMX_ErrorNone
Loop for number of output buffers, which is retrieved earlier using OMX_GetParameter with OMX_IndexParamPortDefinition
(7) OMX_FreeBuffer(handle, OUTPUT_PORT_INDEX, bufferPtr)
(8) Return OMX_ErrorNone
(9) EventHandler(OMX_EventCmdComplete,OMX_StateLoaded) notify client that components is moved to LOADED state from IDLE
(10) OMX_FreeHandle(handle)
(11) Deinitialize and unload the dynamic library
(12) Return OMX_ErrorNone
(13) OMX_DeInit
(14) Return OMX_ErrorNone

Figure 3-17 Sequence diagram for deinitialization

The numbers in parentheses in Figure 3-17 refer to the following steps:

1. Move the component from the Execute state to the Idle state (1).
2. Wait for all buffers to be returned by the component.

3. The component returns all buffers to the IL client (2).
4. The component generates `OMX_EventCmdComplete` for the Execute→Idle state transition (3).
5. Transition the component from the Idle state to the Loaded state (4).
6. Free all input and output buffers (5 to 8).
7. Wait for `OMX_EventCmdComplete` for the Idle→Loaded state transition.
8. The client receives the command `OMX_EventCmdComplete` for the Loaded-to-Idle state transition (9).
9. Call `OMX_FreeHandle` to the OMX core to release the component handle (10).
10. Call `OMX_DeInit` to deinitialize the OMX core (13).

3.13 Guidelines for enabling B frames and MetaBuffer mode

3.13.1 Enabling B frames

Video encoding with B frames is available only on certain codec/profile configurations. B frames are available by default when a profile is higher than:

- OMX_VIDEO_MPEG4ProfileSimple for MPEG4 encoding
- OMX_VIDEO_AVCProfileBaseline for AVC encoding

`OMXCodec` sets the B frames by calling `OMX_SetParameter()` to the OMX IL component with a codec type of `OMX_IndexParamVideoMpeg4` or `OMX_IndexParamvideoAvc` along the `nBFrames` parameter, which specifies the number of B frames within the GOP structure.

3.13.2 MetaBuffer mode

3.13.2.1 Initiation of Metadata mode

Call `OMX_SetParameter` with an extension of `OMX_QcomIndexParamVideoEncodeMetaBufferMode`. This enables the component to operate in MetaBuffer mode where the `pBuffer` pointer in the OpenMAX buffer header contains the metadata instead of the actual buffer containing YUV. QTI defines the metastructure as:

```
typedef enum {
kMetadataBufferTypeCameraSource = 0,
kMetadataBufferTypeGrallocSource = 1,
} MetadataBufferType;
typedef struct buffer_handle_t
{
    int version;        /* sizeof(native_handle_t) */
    int numFds;         /* number of file-descriptors at &data[0] */
    int numInts;        /* number of ints at &data[numFds] */
    int data[0];        /* numFds + numInts ints */
} buffer_handle_t;
```

```
typedef struct encoder_media_buffer_type {
MetadataBufferType buffer_type;
buffer_handle_t meta_handle;
} encoder_media_buffer_type;
```

Data from `buffer_handle_t` should be populated as:

```
Input_pmem_info.fd = media_buffer->meta_handle->data[0];
Input_pmem_info.offset = media_buffer->meta_handle->data[1];
Input_pmem_info.size = media_buffer->meta_handle->data[2];”
```

3.13.2.2 Loaded to Idle transition

The OMX IL queries the component port definition and the component returns `nBufferSize` as the size of the metamode structure and minimum and maximum buffer count.

The IL client calls the allocate buffer on the encoder input port where the component allocates the buffer header and buffer pointer. This is only the metabuffer `encoder_media_buffer_type` (8 bytes described above).

3.13.2.3 During the Executing state

The IL client must populate this metadata structure with an ION file descriptor, etc., and call `OMX_EmptyThisBuffer`.

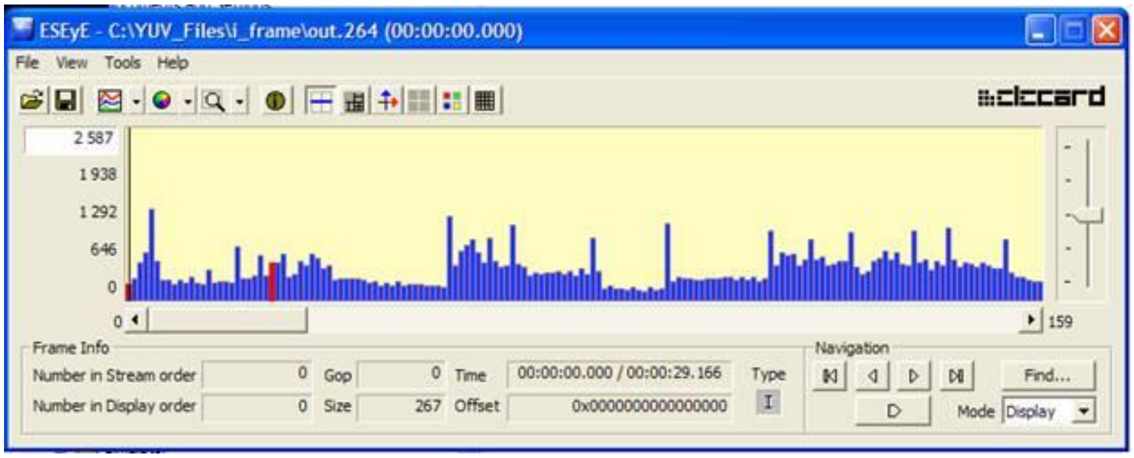
4 IOMX Interface

IOMX is an interface that comes with Froyo’s Stagefright to expose OMX functionalities through the media service server.

Apps that use IOMX to gain access to hardware encoder services:

- Have great flexibility to configure OMX encoder parameters
- Do not have to generate a complete encoder graph using this interface (source, encoder, sink); only an encoding block is created
- Root privileges are not necessary

As illustrated in Figure 4-1, frame request (red bar) and bitrate (bar sizes) can be verified using a bitstream analyzer (Elecard StreamEye shown).



ESEyE - C:\YUV_Files\i_frame\out.264 (00:00:00.000)

Frame Info

Number in Stream order 0 Gop 0 Time 00:00:00.000 / 00:00:29.166 Type I

Number in Display order 0 Size 267 Offset 0x0000000000000000

Mode Display

Figure 4-1 Frame request (red bar) and bitrate (bar sizes) on a bitstream analyzer

5 Limitations

5.1 Generic limitations for encoders

Some generic limitations of encoders are:

- The OMX core and codec interfaces are OMX 1.1-based.
- The OMX core and codec interfaces are *not* backward-compatible with the OMX 1.0 core and OMX 1.0 IL.

EXHIBIT 1

PLEASE READ THIS LICENSE AGREEMENT ("AGREEMENT") CAREFULLY. THIS AGREEMENT IS A BINDING LEGAL AGREEMENT ENTERED INTO BY AND BETWEEN YOU (OR IF YOU ARE ENTERING INTO THIS AGREEMENT ON BEHALF OF AN ENTITY, THEN THE ENTITY THAT YOU REPRESENT) AND QUALCOMM TECHNOLOGIES, INC. ("QTI" "WE" "OUR" OR "US"). THIS IS THE AGREEMENT THAT APPLIES TO YOUR USE OF THE DESIGNATED AND/OR ATTACHED DOCUMENTATION AND ANY UPDATES OR IMPROVEMENTS THEREOF (COLLECTIVELY, "MATERIALS"). BY USING OR COMPLETING THE INSTALLATION OF THE MATERIALS, YOU ARE ACCEPTING THIS AGREEMENT AND YOU AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO THESE TERMS, QTI IS UNWILLING TO AND DOES NOT LICENSE THE MATERIALS TO YOU. IF YOU DO NOT AGREE TO THESE TERMS YOU MUST DISCONTINUE AND YOU MAY NOT USE THE MATERIALS OR RETAIN ANY COPIES OF THE MATERIALS. ANY USE OR POSSESSION OF THE MATERIALS BY YOU IS SUBJECT TO THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT.

1.1 **License.** Subject to the terms and conditions of this Agreement, including, without limitation, the restrictions, conditions, limitations and exclusions set forth in this Agreement, Qualcomm Technologies, Inc. ("QTI") hereby grants to you a nonexclusive, limited license under QTI's copyrights to use the attached Materials; and to reproduce and redistribute a reasonable number of copies of the Materials. You may not use Qualcomm Technologies or its affiliates or subsidiaries name, logo or trademarks; and copyright, trademark, patent and any other notices that appear on the Materials may not be removed or obscured. QTI shall be free to use suggestions, feedback or other information received from You, without obligation of any kind to You. QTI may immediately terminate this Agreement upon your breach. Upon termination of this Agreement, Sections 1.2-4 shall survive.

1.2 **Indemnification.** You agree to indemnify and hold harmless QTI and its officers, directors, employees and successors and assigns against any and all third party claims, demands, causes of action, losses, liabilities, damages, costs and expenses, incurred by QTI (including but not limited to costs of defense, investigation and reasonable attorney's fees) arising out of, resulting from or related to: (i) any breach of this Agreement by You; and (ii) your acts, omissions, products and services. If requested by QTI, You agree to defend QTI in connection with any third party claims, demands, or causes of action resulting from, arising out of or in connection with any of the foregoing.

1.3 **Ownership.** QTI (or its licensors) shall retain title and all ownership rights in and to the Materials and all copies thereof, and nothing herein shall be deemed to grant any right to You under any of QTI's or its affiliates' patents. You shall not subject the Materials to any third party license terms (e.g., open source license terms). You shall not use the Materials for the purpose of identifying or providing evidence to support any potential patent infringement claim against QTI, its affiliates, or any of QTI's or QTI's affiliates' suppliers and/or direct or indirect customers. QTI hereby reserves all rights not expressly granted herein.

1.4 **WARRANTY DISCLAIMER.** YOU EXPRESSLY ACKNOWLEDGE AND AGREE THAT THE USE OF THE MATERIALS IS AT YOUR SOLE RISK. THE MATERIALS AND TECHNICAL SUPPORT, IF ANY, ARE PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED. QTI ITS LICENSORS AND AFFILIATES MAKE NO WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THE MATERIALS OR ANY OTHER INFORMATION OR DOCUMENTATION PROVIDED UNDER THIS AGREEMENT, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR AGAINST INFRINGEMENT, OR ANY EXPRESS OR IMPLIED WARRANTY ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. NOTHING CONTAINED IN THIS AGREEMENT SHALL BE CONSTRUED AS (I) A WARRANTY OR REPRESENTATION BY QTI, ITS LICENSORS OR AFFILIATES AS TO THE VALIDITY OR SCOPE OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT OR (II) A WARRANTY OR REPRESENTATION BY QTI THAT ANY MANUFACTURE OR USE WILL BE FREE FROM INFRINGEMENT OF PATENTS, COPYRIGHTS OR OTHER INTELLECTUAL PROPERTY RIGHTS OF OTHERS, AND IT SHALL BE THE SOLE RESPONSIBILITY OF YOU TO MAKE SUCH DETERMINATION AS IS NECESSARY WITH RESPECT TO THE ACQUISITION OF LICENSES UNDER PATENTS AND OTHER INTELLECTUAL PROPERTY OF THIRD PARTIES.

1.5 **LIMITATION OF LIABILITY.** IN NO EVENT SHALL QTI, QTI'S AFFILIATES OR ITS LICENSORS BE LIABLE TO YOU FOR ANY INCIDENTAL, CONSEQUENTIAL OR SPECIAL DAMAGES, INCLUDING BUT NOT LIMITED TO ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL DAMAGES, ARISING OUT OF THE USE OR INABILITY TO USE, OR THE DELIVERY OR FAILURE TO DELIVER, ANY OF THE MATERIALS, OR ANY BREACH OF ANY OBLIGATION UNDER THIS AGREEMENT, EVEN IF QTI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE FOREGOING LIMITATION OF LIABILITY SHALL REMAIN IN FULL FORCE AND EFFECT REGARDLESS OF WHETHER YOUR REMEDIES HEREUNDER ARE DETERMINED TO HAVE FAILED OF THEIR ESSENTIAL PURPOSE. THE ENTIRE LIABILITY OF QTI, QTI's AFFILIATES AND ITS LICENSORS, AND THE SOLE AND EXCLUSIVE REMEDY OF YOU, FOR ANY CLAIM OR CAUSE OF ACTION ARISING HEREUNDER (WHETHER IN CONTRACT, TORT, OR OTHERWISE) SHALL NOT EXCEED US$10.

2. **COMPLIANCE WITH LAWS; APPLICABLE LAW.** You agree to comply with all applicable local, international and national laws and regulations and with U.S. Export Administration Regulations, as they apply to the subject matter of this Agreement. This Agreement is governed by the laws of the State of California, excluding California's choice of law rules.

3. **CONTRACTING PARTIES.** If the Materials are downloaded on any computer owned by a corporation or other legal entity, then this Agreement is formed by and between QTI and such entity. The individual accepting the terms of this Agreement represents and warrants to QTI that they have the authority to bind such entity to the terms and conditions of this Agreement.

4. **MISCELLANEOUS PROVISIONS.** This Agreement, together with all exhibits attached hereto, which are incorporated herein by this reference, constitutes the entire agreement between QTI and You and supersedes all prior negotiations, representations and agreements between the parties with respect to the subject matter hereof. No addition or modification of this Agreement shall be effective unless made in writing and signed by the respective representatives of QTI and You. The restrictions, limitations, exclusions and conditions set forth in this Agreement shall apply even if QTI or any of its affiliates becomes aware of or fails to act in a manner to address any violation or failure to comply therewith. You hereby acknowledge and agree that the restrictions, limitations, conditions and exclusions imposed in this Agreement on the rights granted in this Agreement are not a derogation of the benefits of such rights. You further acknowledges that, in the absence of such restrictions, limitations, conditions and exclusions, QTI would not have entered into this Agreement with You. Each party shall be responsible for and shall bear its own expenses in connection with this Agreement. If any of the provisions of this Agreement are determined to be invalid, illegal, or otherwise unenforceable, the remaining provisions shall remain in full force and effect. This Agreement is entered into solely in the English language, and if for any reason any other language version is prepared by any party, it shall be solely for convenience and the English version shall govern and control all aspects. If You are located in the province of Quebec, Canada, the following applies: The Parties hereby confirm they have requested this Agreement and all related documents be prepared in English.