Jacinto6 Android Video Encoder Software Design Specification

User's Guide

Literature Number: SPRUI39
April 2016

Contents

List of Figures

List of Tables

User's Guide
SPRUI39–April 2016

Jacinto6 Android Video Encoder Software Design Specification

1 Introduction

1.1 Jacinto6 Hardware for Video Encoding

Jacinto6 hardware supports image and video accelerator high definition (IVA - HD) to handle complex video codecs with guaranteed power and performance.

The video buffers are allocated through DMM TILER, which arranges the pixels in the predefined tiles. This is an important hardware feature to improve macro block fetch latency and to achieve image rotation efficiently.

The IVA-HD accelerator is controlled from the image processing unit (IPU) M4 to ensure real-time data processing, and MPU (ARM cortex A15) is dedicated for HLOS.

The software components span across multiple cores to parallelize the processing and achieve the best quality video encoding.

Figure 1 shows the hardware blocks involved in video encoding.

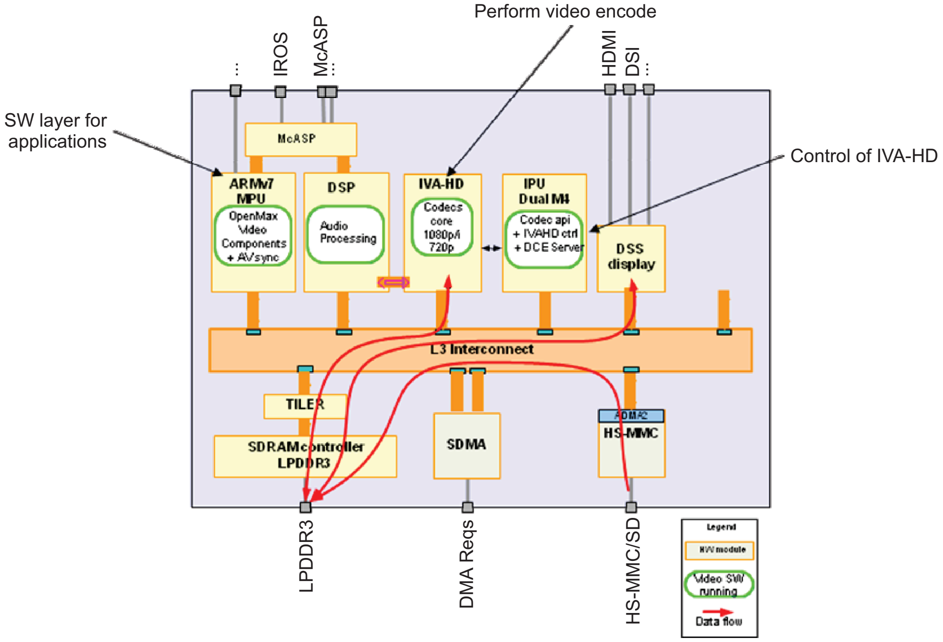

Figure 1. Jacinto6 Hardware Accelerators for Video Encoding

Android is a trademark of Google Inc.
All other trademarks are the property of their respective owners.

Acronyms and Definitions

Table 1. Acronym Definitions

Acronym	Definition
API	Application programming interface
CE	Codec engine
DCE	Distributed codec engine
DSS	Display subsystem on Jacinto6
FC	Framework component
HAL	Hardware abstraction layer
HW	Hardware
HWC	Hardware composer
IPC	Inter-processor communication
IVAHD	Image video accelerator for HD
OMX	http://en.wikipedia.org/wiki/OpenMAX OpenMax standard for multimedia
PVR	Refers to the PowerVR technologies and software provided by Imagination Technologies
SGX	The graphics IP provided by Imagination Technologies
SW	Software

1.2 Multicore Responsibilities During Video Encoding Usecase

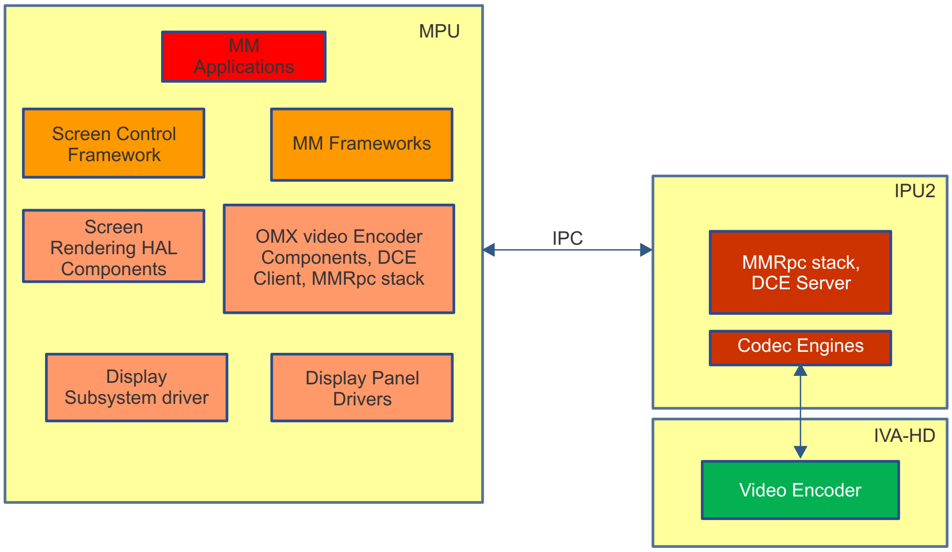

Figure 2. Multicore Responsibilities Video Encoding Usecase

1.2.1 MPU (Cortex A15)

- Executes Android™ multimedia stack, stagefright engine and Mpeg4 writer
- Manages OMX state machine for video encoder component
- Interacts with IPU core through IPC and DCE interface for video encoding

1.2.2 IPU (M4)

- Executes DCE server, framework components, and codec engine
- Manages IVA-HD configuration, and handles IVA-HD interrupts and messages
- Interacts with IVA-HD accelerators for video encoding through the codec engine interface

1.2.3 IVA-HD

- Executes video encoder algorithm

1.3 Android MultiMedia Stack

The vanilla Android multimedia stack provides command line test (recordvideo) to verify video encoding from file to file. This test case reads raw .yuv frames from input file, encodes them using stagefright MediaCodec interface, and writes the encoded bit stream into a .mp4 container.

Video encoding is accelerated with IVA-HD codecs integrated into Android via OMX Core, OMXComponents, DCE and IPC components. Figure 3 shows the mapping of multicore responsibilities to the Android multimedia stack.

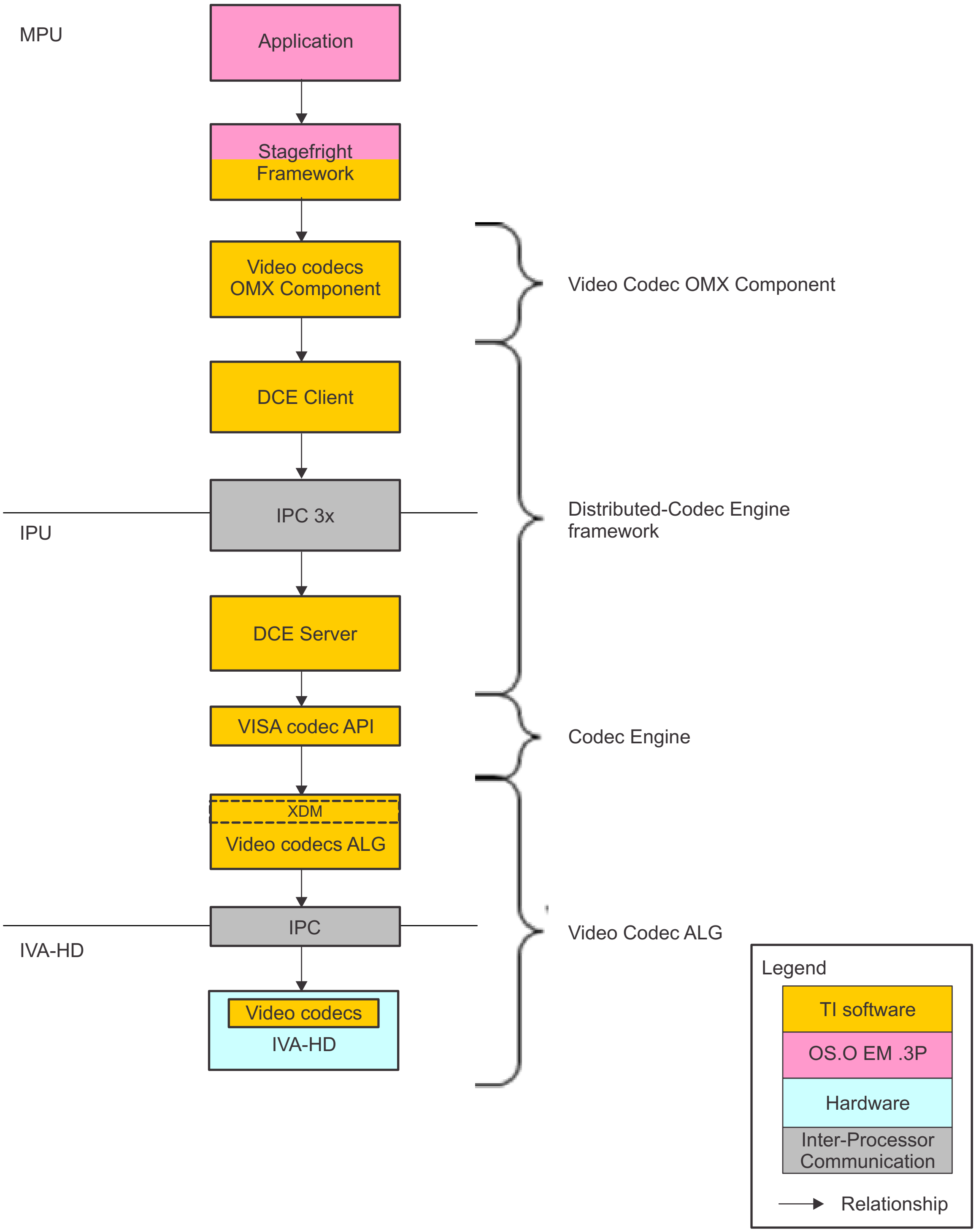

Figure 3. Android Multimedia Stack vs Multicore Mapping

Stagefright framework is the concrete framework with media writer, OMX client classes. It has IOMX binder interface to integrate HW accelerated codecs as OMX components.

The DCE client-on-host (MPU) provides hardware abstraction for the IPC and remote core configuration and message passing. DCE interacts with IPC via MMRPC component and provides simple create/delete/process interface for the OMX components.

IPC provides a complete stack for message creation and passing between the cores.

The DCE server on the remote core handles the client messages, and configures and controls IVA-HD accelerators.

Codec algorithm runs on IVA-HD and operates in frame mode.

2 Functional Overview

This section provides the detailed description about the Android multimedia software components in a top-down approach.

2.1 Android Video Encoder Architecture

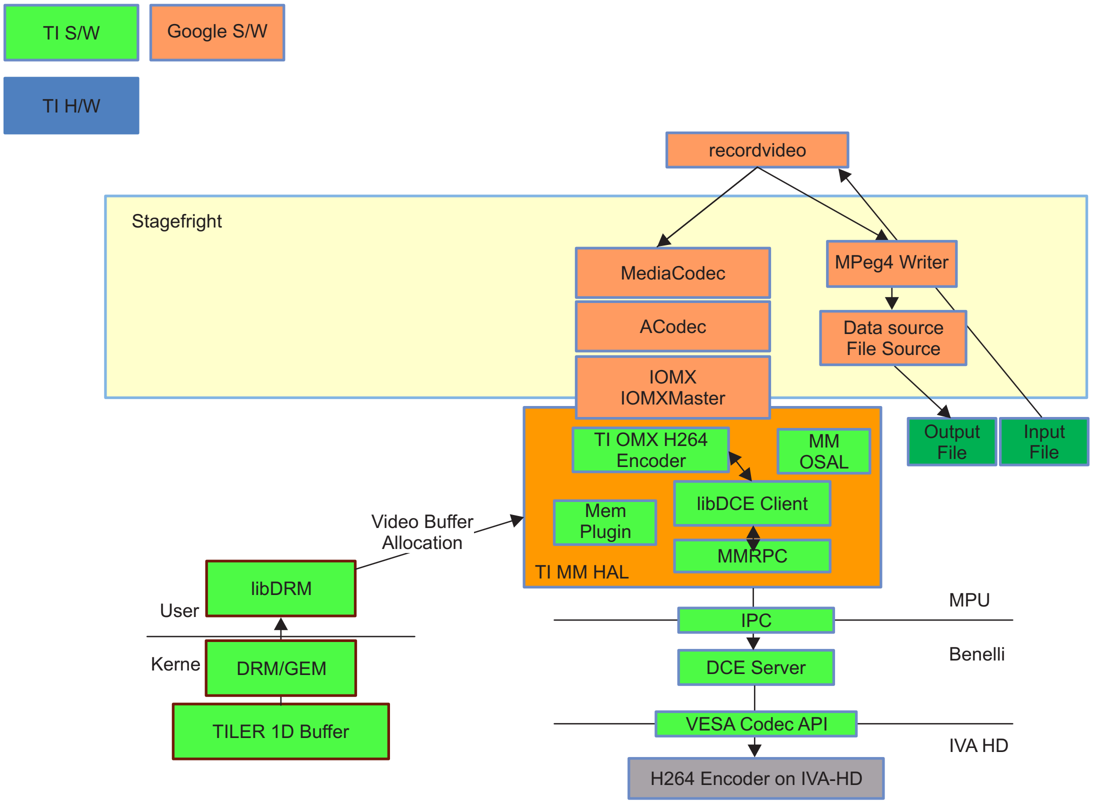

Figure 4. Android Encoder Video Architecture

2.1.1 StageFright

The StageFright framework is a collection of the basic media components like extractors, writers, OMX client, and OMX plugin interface.

MediaWriter is responsible for creating the MP4 container from the encoded bit-stream.

MediaSource (MediaCodec/OMXCodec) acts as OMX Client and interacts with the OMX components through IOMX binder interface. OMX plugin interface provides hooks for vendor specific OMX Core registration and hence the OMX component plugin.

MediaCodec.xml provides a means to select the appropriate codecs based on device capabilities.

2.2 TI MM HAL

TI MM HAL implements OMX core, OMX components, OSAL for OS primitives (mutex, semaphore, pipes, events), and the DCE client.

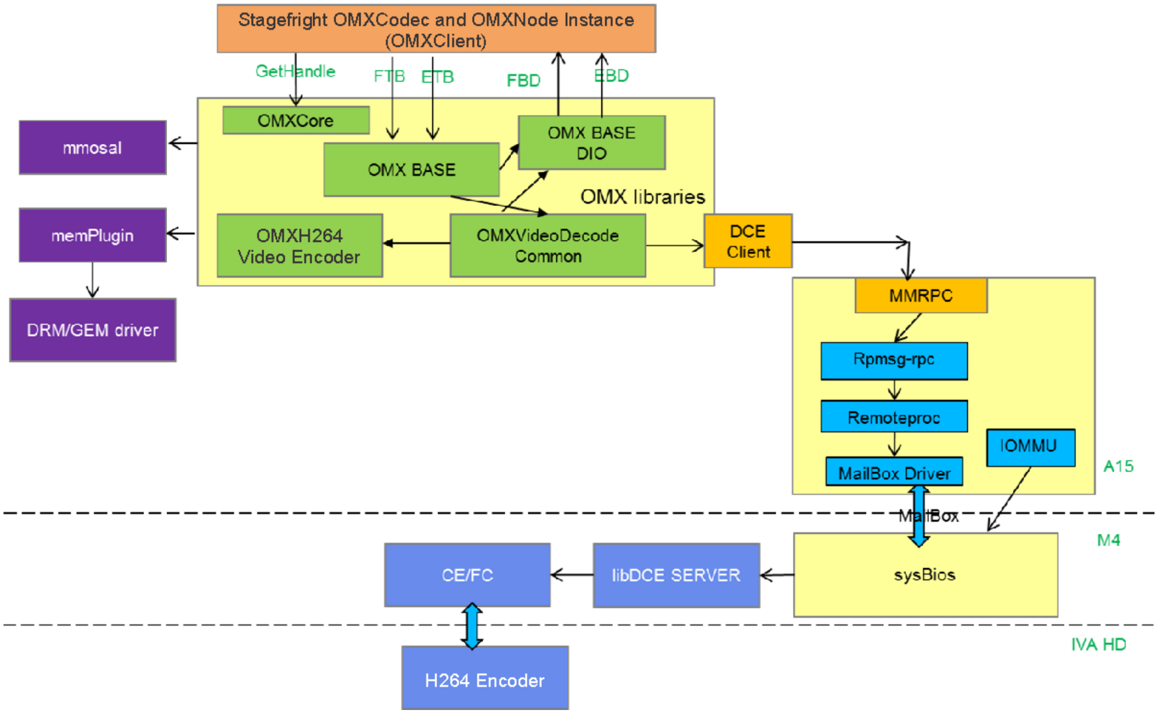

Figure 5. TI MM HAL Components

IVA-HD hardware-accelerated video encoders are integrated into Android as OMX components.

As shown in Figure 5, the core encoder algorithm runs on IVA-HD. IPU (M4) implements codec engine (CE) and framework component (FC) to load and control the codec. DCE server is a wrapper on top of CE/FC, and handles the client requests for bit-stream processing.

The OMX components on the host side implement the complete OMX state machine for getHandle, set/get Param, set/get Config, FillThisBuffer, EmptyThisBbuffer, FillBufferDone, and EmptyBufferDone. HAL also implements memallocator for parameter buffers, and OSAL for OS primitives such as pipes, mutex, and so forth. DCE client on the host side provides a simple interface for the OMX components to load and configure the encoder and then process the frames.

The OMX framework is designed to split the functionality into logical blocks: OMXBase, Data I/O (DIO), video encoder common, and video encoder-specific modules. The OMX base implements the base class for the OMX state machine, reused among multiple encoders and decoders. It provides buffer allocation, message passing through pipes, and port handling through DIO. The OMX video encoder common is derived from the OMXBase, and overrides some of the OMX methods to handle encoder-specific modules. OMX video encoder component implements setparam/getparam, setconfig/getconfig specific to H264 encoder and the buffer management for both metadata and non-meta data modes.

The IPC between IPU (M4) and MPU (A15) follows the rpmsg/rpc protocol. On Android, use an MMRPC interface to rpmsg-rpc, remoteproc and mailbox drivers.

DCE client implements VIDENC2 interface defined in CE. This is a simple interface with create, control, process, process_async , process_wait and delete methods to talk to codec algorithm. Current software implementation supports only the synchronous functionality, and doesn’t support process_async and process_wait apis from the CE standard interface. This means that we can support only frame level encoding.

2.3 Buffer Management

OMX video encoder component supports two modes of operation defined in Android: meta-data mode and non-meta-data mode

2.3.1 Meta-Data Mode

When the H264 encoder component is configured for meta-data mode, the i/p port carries only the meta-data for the i/p buffers. The meta-data carries the information about the i/p data type (whether it is a GRALLOC native handle or virtual pointer), and the handle for the i/p buffer.

Since the buffers are allocated by HAL components, the actual buffer pointer/fd can be extracted by talking to the buffer allocator in a custom way.

This mode is very efficient as it involves moving only the meta-data across process boundaries and it avoids the actual buffer copies.

This mode also provides flexibility to support custom pixel formats for encoder without changing Android native stack; this can be achieved by configuring the component for Android_Opaqueue_format and extracting the actual color format from the buffer handle.

2.3.2 Non-Meta-Data Mode

When the H264 encoder component is configured for non-meta data mode, i/p port carries the actual buffer pointer for i/p buffer. During the IOMX process boundary switch, the client buffer gets copied into the OMX component buffer if the component and client doesn’t live in the same process. This method is not recommended for high performance use cases because of memcpy overhead.

Codec output buffers are the bit-stream buffers and are allocated by the OMX component through libDRM library.

2.3.3 Buffer Flow

The encoder module is integrated as the OMX Component, hence the buffer flow follows the OMX standard calls: FillThisBuffer(FTB), EmtpyThisBuffer(ETB), FillBufferDone(FBD) and EmptyBufferDone(EBD).

3 Interfaces

3.1 HAL Interfaces

3.1.1 OMXBase Structure

```
typedef struct OMXBaseComp
{
    OMX_STRING                  cComponentName;
    OMX_VERSIONTYPE             nComponentVersion;
    OMX_PORT_PARAM_TYPE         *pAudioPortParams;
    OMX_PORT_PARAM_TYPE         *pVideoPortParams;
    OMX_PORT_PARAM_TYPE         *pImagePortParams;
    OMX_PORT_PARAM_TYPE         *pOtherPortParams;
    OMX_U32                     nNumPorts;
    OMX_U32                     nMinStartPortIndex;
    OMXBase_Port                **pPorts;
    OMX_BOOL                    bNotifyForAnyPort;
    OMXBaseComp_Pvt             *pPvtData;
    OMX_STATETYPE               tCurState;
    OMX_STATETYPE               tNewState;
    OMX_PTR                     pMutex;

    OMX_ERRORTYPE (*fpCommandNotify)(OMX_HANDLETYPE hComponent, OMX_COMMANDTYPE Cmd,
                                OMX_U32 nParam, OMX_PTR pCmdData);

    OMX_ERRORTYPE (*fpDataNotify)(OMX_HANDLETYPE hComponent);


    OMX_ERRORTYPE (*fpReturnEventNotify)(OMX_HANDLETYPE hComponent, OMX_EVENTTYPE eEvent,
                                OMX_U32 nEventData1, OMX_U32 nEventData2, OMX_PTR pEventData);

    OMX_ERRORTYPE (*fpXlateBuffHandle)(OMX_HANDLETYPE hComponent, OMX_PTR pBufferHdr, OMX_BOOL
bRegister);

}OMXBaseComp;
```

3.1.2 OMXBase DIO Structure

```
typedef struct OMX_DIO_Object {

    OMX_PTR pContext;

    OMX_ERRORTYPE (*open)(OMX_HANDLETYPE handle,
                          OMX_DIO_OpenParams *pParams);

    OMX_ERRORTYPE (*close)(OMX_HANDLETYPE handle);

    OMX_ERRORTYPE (*queue)(OMX_HANDLETYPE handle,
                           OMX_PTR pBuffHeader);

    OMX_ERRORTYPE (*dequeue)(OMX_HANDLETYPE handle,
                             OMX_PTR *pBuffHeader);

    OMX_ERRORTYPE (*send)(OMX_HANDLETYPE handle,
                          OMX_PTR pBuffHeader);

    OMX_ERRORTYPE (*cancel)(OMX_HANDLETYPE handle,
                            OMX_PTR pBuffHeader);

    OMX_ERRORTYPE (*control)(OMX_HANDLETYPE handle,
                             OMX_DIO_CtrlCmdType nCmdType,
                             OMX_PTR pParams);

    OMX_ERRORTYPE (*getcount)(OMX_HANDLETYPE handle,
                              OMX_U32 *pCount);

    OMX_ERRORTYPE (*deinit)(OMX_HANDLETYPE handle);

    OMX_BOOL bOpened;

}OMX_DIO_Object;
```

3.1.3 OMXVideoEncoder Structure

```
typedef struct OMXH264VideoEncoderComponent {
    /* base component handle */
    OMXBaseComp         sBase;

    /* codec and engine handles */
    Engine_Handle       pCEhandle;
    Engine_Error        tCEerror;
    VIDENC2_Handle      pVidEncHandle;
    OMX_BOOL            bCodecCreate;
    OMX_BOOL            bCodecCreateSettingsChange;

    /* Encoder static/dynamic/buf args */
    IH264ENC_Params         *pVidEncStaticParams;
    IH264ENC_DynamicParams  *pVidEncDynamicParams;
    IH264ENC_Status         *pVidEncStatus;
    IH264ENC_InArgs         *pVidEncInArgs;
    IH264ENC_OutArgs        *pVidEncOutArgs;
    IVIDEO2_BufDesc         *pVedEncInBufs;
    XDM2_BufDesc            *pVedEncOutBufs;

    /* omx component statemachine variables */
    OMX_BOOL                bInputPortDisable;
    OMX_BOOL                bCodecFlush;
    PARAMS_UPDATE_STATUS    bCallxDMSetParams;
    OMX_BOOL                bAfterEOSReception;
    OMX_BOOL                bNotifyEOSEventToClient;
    OMX_BOOL                bPropagateEOSToOutputBuffer;
    OMX_BOOL                bSetParamInputIsDone;

    /* codec config handling variables*/
    OMXBase_CodecConfigBuf  sCodecConfigData;
    OMX_BOOL                bSendCodecConfig;
    OMX_U32                 nCodecConfigSize;
    OMX_BOOL                bAfterGenHeader;

    /* internal buffer tracking arrays */
    OMX_BUFFERHEADERTYPE    **pCodecInBufferArray;
    OMXBase_BufHdrPvtData   *pCodecInBufferBackupArray;

    /* temporary memory to meet and codec and dce requirements */
    MemHeader               *pTempBuffer[2];

    OMX_BOOL                bInputMetaDataBufferMode;
    OMX_PTR                 hCC;
    IMG_native_handle_t     **pBackupBuffers;
    alloc_device_t          *mAllocDev;

} OMXH264VidEncComp;
```

3.1.4 DCE Client Interface (VIDENC2 Interface)

```
*
 *  ======== VIDENC2_control ========
 */
/**
 *  @brief      Execute the control() method in this instance of a video
 *              encoder algorithm.
 *
 *  @param[in]  handle  Handle to a created video encoder instance.
 *  @param[in]  id      Command id for XDM control operation.
 *  @param[in]  params  Runtime control parameters used for encoding.
 *  @param[out] status  Status info upon completion of encode operation.
 *
 *  @pre        @c handle is a valid (non-NULL) video encoder handle
 *              and the video encoder is in the created state.
 *
 *  @retval     #VIDENC2_EOK         Success.
 *  @retval     #VIDENC2_EFAIL       Failure.
 *  @retval     #VIDENC2_EUNSUPPORTED    The requested operation
 *                                       is not supported.
 *
 *  @remark     This is a blocking call, and will return after the control
 *              command has been executed.
 *
 *  @remark     If an error is returned, @c status->extendedError may
 *              indicate further details about the error.  See
 *              #VIDENC2_Status::extendedError for details.
 *
 *  @sa         VIDENC2_create()
 *  @sa         VIDENC2_delete()
 *  @sa         IVIDENC2_Fxns::process()
 */
extern Int32 VIDENC2_control(VIDENC2_Handle handle, VIDENC2_Cmd id,
        VIDENC2_DynamicParams *params, VIDENC2_Status *status);


/*
 *  ======== VIDENC2_create ========
 */
/**
```

```
 *  @brief      Create an instance of a video encoder algorithm.
 *
 *  Instance handles must not be concurrently accessed by multiple threads;
 *  each thread must either obtain its own handle (via VIDENC2_create()) or
 *  explicitly serialize access to a shared handle.
 *
 *  @param[in]  e       Handle to an opened engine.
 *  @param[in]  name    String identifier of the type of video encoder
 *                      to create.
 *  @param[in]  params  Creation parameters.
 *
 *  @retval     NULL            An error has occurred.
 *  @retval     non-NULL        The handle to the newly created video encoder
 *                              instance.
 *
 *  @remarks    @c params is optional.  If it's not supplied, codec-specific
 *              default params will be used.
 *
 *  @remark     Depending on the configuration of the engine opened, this
 *              call may create a local or remote instance of the video
 *              encoder.
 *
 *  @codecNameRemark
 *
 *  @sa         Engine_open()
 *  @sa         VIDENC2_delete()
 */
extern VIDENC2_Handle VIDENC2_create(Engine_Handle e, String name,
    VIDENC2_Params *params);


/*
 *  ======== VIDENC2_delete ========
 */
/**
 *  @brief      Delete the instance of a video encoder algorithm.
 *
 *  @param[in]  handle  Handle to a created video encoder instance.
 *
 *  @remark     Depending on the configuration of the engine opened, this
 *              call may delete a local or remote instance of the video
 *              encoder.
 *
 *  @pre        @c handle is a valid (non-NULL) handle which is
 *              in the created state.
 *
 *  @post       All resources allocated as part of the VIDENC2_create()
 *              operation (memory, DMA channels, etc.) are freed.
 *
 *  @sa         VIDENC2_create()
 */
extern Void VIDENC2_delete(VIDENC2_Handle handle);


/*
 *  ======== VIDENC2_process ========
 */
/**
 *  @brief      Execute the process() method in this instance of a video
 *              encoder algorithm.
 *
 *  @param[in]  handle  Handle to a created video encoder instance.
 *  @param[in]  inBufs  A buffer descriptor containing input buffers.
 *  @param[out] outBufs A buffer descriptor containing output buffers.
 *  @param[in]  inArgs  Input Arguments.
 *  @param[out] outArgs Output Arguments.
 *
 *  @pre        @c handle is a valid (non-NULL) video encoder handle
 *              and the video encoder is in the created state.
 *
 *  @retval     #VIDENC2_EOK         Success.
 *  @retval     #VIDENC2_EFAIL       Failure.
 *  @retval     #VIDENC2_EUNSUPPORTED    The requested operation
 *                                       is not supported.
 *
 *  @remark     Since the VIDENC2 decoder contains support for asynchronous
 *              buffer submission and retrieval, this API becomes known as
 *              synchronous in nature.
 *
 *  @remark     This is a blocking call, and will return after the data
 *              has been encoded.
 *
 *  @remark     The buffers supplied to VIDENC2_process() may have constraints
 *              put on them.  For example, in dual-processor, shared memory
 *              architectures, where the codec is running on a remote
 *              processor, the buffers may need to be physically contiguous.
 *              Additionally, the remote processor may place restrictions on
 *              buffer alignment.
 *
 *  @remark     If an error is returned, @c outArgs->extendedError may
 *              indicate further details about the error.  See
 *              #VIDENC2_OutArgs::extendedError for details.
 *
 *  @sa         VIDENC2_create()
 *  @sa         VIDENC2_delete()
 *  @sa         VIDENC2_control()
 *  @sa         VIDENC2_processAsync()
 *  @sa         VIDENC2_processWait()
 *  @sa         IVIDENC2_Fxns::process() - the reflected algorithm interface,
 *                                         which may contain further usage
 *                                         details.
 */
extern Int32 VIDENC2_process(VIDENC2_Handle handle, IVIDEO2_BufDesc *inBufs,
        XDM2_BufDesc *outBufs, VIDENC2_InArgs *inArgs,
        VIDENC2_OutArgs *outArgs);
```

3.1.5 MmRpc Interface

```
*!
 *  @brief      Invoke a remote procedure call
 *
 *  @param[in]      handle  MmRpc handle, obtained from MmRpc_create()
 *  @param[in]      ctx     Context with which to invoke the remote service
 *  @param[in, out] ret     Return value from the remotely invoked service
 *
 *  @sa MmRpc_create()
 *  @sa MmRpc_delete()
 */
int MmRpc_call(MmRpc_Handle handle, MmRpc_FxnCtx *ctx, int32_t *ret);

/*!
 *  @brief      Create an MmRpc instance
 *
 *  @param[in]      service     Name of the service to create
 *  @param[in]      params      Initialized MmRpc parameters
 *  @param[in,out]  handlePtr   Space to hold the MmRpc handle
 *
 *  @retval     MmRpc_S_SUCCESS @copydoc MmRpc_S_SUCCESS
 *  @retval     MmRpc_E_FAIL    @copydoc MmRpc_E_FAIL
 *
 *  @remark     This instantiates an instance of the service on a remote
 *              core.  Each remote instance consists of a unique thread
 *              listening for requests made via a call to MmRpc_call().
 */
int MmRpc_create(const char *service, const MmRpc_Params *params,
        MmRpc_Handle *handlePtr);

/*!
 *  @brief      Delete an MmRpc instance
 *
 *  @param[in]  handlePtr  MmRpc handle, obtained from MmRpc_create()
 *
 *  @sa MmRpc_create()
 */
int MmRpc_delete(MmRpc_Handle *handlePtr);

/*!
 *  @brief      Release buffers which were declared in use
 *
 *  @param[in]  handle  Service handle returned by MmRpc_create()
 *  @param[in]  type    Buffer descriptor type
 *  @param[in]  num     Number of elements in @c desc array
 *  @param[in]  desc    Pointer to array of buffer descriptors
 *
 *  @remark     When the remote processor no longer needs a reference
 *              to a buffer, calling MmRpc_release() will release the
 *              buffer and any associated resources.
 *
 *  @retval     MmRpc_S_SUCCESS         @copydoc MmRpc_S_SUCCESS
 *  @retval     MmRpc_E_INVALIDPARAM    @copydoc MmRpc_E_INVALIDPARAM
 *  @retval     MmRpc_E_NOMEM           @copydoc MmRpc_E_NOMEM
 *  @retval     MmRpc_E_SYS             @copydoc MmRpc_E_SYS
 *
 *  @sa         MmRpc_use()
 */
int MmRpc_release(MmRpc_Handle handle, MmRpc_BufType type, int num,
        MmRpc_BufDesc *desc);

/*!
 *  @brief      Declare the use of the given buffers
 *
 *  @param[in]  handle  Service handle returned by MmRpc_create()
 *  @param[in]  type    Buffer descriptor type
 *  @param[in]  num     Number of elements in @c desc array
 *  @param[in]  desc    Pointer to array of buffer descriptors
 * *  @remark     When using MmRpc_call() to invoke remote function calls,
 *              any referenced buffers will be made available to the
 *              remote processor only for the duration of the remote
 *              function call. If the remote processor maintains a
 *              reference to the buffer across multiple invocations of
 *              MmRpc_call(), then the application must declare the buffer
 *              "in use". This will make the buffer persistent.
 *
 *  @remark     The application must release the buffer when it is no
 *              longer needed.
 *
 *  @code
 *      #include <ti/ipc/mm/MmRpc.h>
 *
 *      MmRpc_BufDesc desc[2];
 *
 *      desc[0].handle = fd1;
 *      desc[1].handle = fd2;
 *
 *      MmRpc_use(h, MmRpc_BufType_Handle, 2, desc);
 *  @endcode
 *
 *  @retval     MmRpc_S_SUCCESS         @copydoc MmRpc_S_SUCCESS
 *  @retval     MmRpc_E_INVALIDPARAM    @copydoc MmRpc_E_INVALIDPARAM
 *  @retval     MmRpc_E_NOMEM           @copydoc MmRpc_E_NOMEM
 *  @retval     MmRpc_E_SYS             @copydoc MmRpc_E_SYS
 *
 *  @sa         MmRpc_release()
 */
int MmRpc_use(MmRpc_Handle handle, MmRpc_BufType type, int num,
        MmRpc_BufDesc *desc);

/*!
 *  @brief      Initialize the instance create parameter structure
 *
 */
void MmRpc_Params_init(MmRpc_Params *params);
```

3.2 MediaCodec Selection xml

```
/<MediaCodecs>
    <Encoders>
        <MediaCodec name="OMX.TI.DUCATI1.VIDEO.H264E" type="video/avc" >
            <Quirk name="requires-allocate-on-input-ports" />
            <Quirk name="requires-allocate-on-output-ports" />
        </MediaCodec>

        <MediaCodec name="OMX.google.amrnb.encoder" type="audio/3gpp" />
        <MediaCodec name="OMX.google.amrwb.encoder" type="audio/amr-wb" />
        <MediaCodec name="OMX.google.aac.encoder" type="audio/mp4a-latm" />
        <MediaCodec name="OMX.google.flac.encoder" type="audio/flac" />
    </Encoders>
</MediaCodecs>
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265