1 Introduction
   1.1 Jacinto6 Hardware for Audio Video Playback ................................................................. 4
   1.2 Acronyms and Definitions ............................................................................................... 5
   1.3 Multicore Responsibilities During AV Playback Use-case .................................................. 5
   1.4 Android MultiMedia Stack ............................................................................................... 6
2 Functional Overview ........................................................................................................... 8
   2.1 Android Video Architecture ............................................................................................. 8
   2.2 TI MM HAL ....................................................................................................................... 8
3 Interfaces .......................................................................................................................... 12
   3.1 HAL Interfaces .................................................................................................................. 12
   3.2 MediaCodec xml for Decoder Selection ............................................................................. 17
   3.3 MediaPlayer .................................................................................................................... 18
List of Figures

1. Jacinto6 Hardware Accelerators for Audio Video Playback .................................................. 4
2. Multicore Responsibilities During AV Playback .................................................................... 5
3. Android Multimedia Stack vs Multicore Mapping ................................................................. 7
4. Android Video Architecture .................................................................................................. 8
5. TI MM HAL Components ....................................................................................................... 9
6. Decoder Buffer Allocation and Dataflow ................................................................................ 10
7. Audio Video Synchronization ............................................................................................... 11

List of Tables

1. Acronym Definitions ................................................................................................................ 5
1 Introduction

1.1 Jacinto6 Hardware for Audio Video Playback

Jacinto6 hardware supports image and video accelerator high definition (IVA - HD) to handle complex video codecs with guaranteed power and performance. The video post-processing, such as color conversion, scaling, and cropping, is handled inside the DSS (display subsystem) hardware.

The video buffers are allocated through DMM TILER, which arranges the pixels in the predefined tiles. This hardware feature improves macro block fetch latency and efficiently achieves image rotation.

The IVA-HD accelerator is controlled from the image processing unit (IPU) to ensure real-time data processing.

The MPU (ARM Cortex® A15) is dedicated for HLOS, and the DSP is for audio post processing.

The software components span across multiple cores to parallelize the processing and achieve the best quality AV-synchronized playback.

Figure 1 shows the hardware blocks involved in AV playback.

![Figure 1. Jacinto6 Hardware Accelerators for Audio Video Playback](image-url)
1.2 Acronyms and Definitions

Table 1. Acronym Definitions

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>CE</td>
<td>Codec engine</td>
</tr>
<tr>
<td>DCE</td>
<td>Distributed codec engine</td>
</tr>
<tr>
<td>DSS</td>
<td>Display subsystem on Jacinto6</td>
</tr>
<tr>
<td>FC</td>
<td>Framework component</td>
</tr>
<tr>
<td>HAL</td>
<td>Hardware abstraction layer</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>HWC</td>
<td>Hardware composer</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter-processor communication</td>
</tr>
<tr>
<td>IVAHD</td>
<td>Image video accelerator for HD</td>
</tr>
<tr>
<td>OMX</td>
<td>OpenMax standard for multimedia</td>
</tr>
<tr>
<td>PVR</td>
<td>Refers to the PowerVR technologies and software provided by Imagination Technologies</td>
</tr>
<tr>
<td>SGX</td>
<td>The graphics IP provided by Imagination Technologies</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
</tbody>
</table>

1.3 Multicore Responsibilities During AV Playback Use case

1.3.1 MPU (Cortex A15)
- Executes Android™ multimedia stack, Media Player/Stagefright Engine, and input parsing.
- Interacts with IPU core through IPC and DCE interface for media decoding.
- Allocates decoder output buffers from TILER through the GRALLOC interface.
- Handles the decoded buffer posting to display through surface flinger/HWComposer.
- Responsible for audio decoding and AV sync.

Figure 2. Multicore Responsibilities During AV Playback
1.3.2 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 decoding through the codec engine interface.

1.3.3 IVA-HD  
- Executes video decoder algorithm.

1.3.4 DSP  
- Responsible for audio signal processing, such as SRC, mixing, EQ, and so forth.  
- Renders data to the external audio codec through the McASP interface.

1.4 Android MultiMedia Stack  
The vanilla Android multimedia stack provides a sample application (Gallery2) and middleware framework for media scanner, parsers, and OMX client with hooks to integrate vendor specific codecs for video and audio. It also has the support for software codecs of a few compression types.

Jacinto6 video architecture implements OMX Core, OMXComponents, IPC, and integrates codecs into the Android stagefright framework. Figure 3 shows the mapping of multicore responsibilities to the Android multimedia stack.
Gallery2 player is the default media player in the Android release. It has basic features such as media scanning, thumbnail viewing, and play/pause/resume/seek controls.

MediaPlayer provides JAVA/JNI/Native interface for the applications to talk to native framework.

Stagefright framework is the concrete framework with media extractor, OMX client, media renderer, and audio renderer classes. It has an IOMX binder interface to integrate hardware-accelerated codecs as OMX components.

Distributed codec engine (DCE) client-on-host (MPU) provides hardware abstraction for the IPC, as well as remote core configuration and message passing. DCE interacts with IPC through the MMRPC component, and provides a simple create/delete/process interface for the OMX components.

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

Distributed codec engine (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 Architecture

2.1.1 Gallery Player

Gallery Player is the default media application from Android software, which provides features such as thumbnail view, photo slideshow, and video playback with start/stop/pause/resume/seek options.

2.1.2 StageFright

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

MediaExtractor is responsible for retrieving track data and the corresponding meta-data from the file or from the http stream.

MediaSource (MediaCodec/OMXCodec) acts as OMX client and interacts with the OMX components through the IOMUX binder interface. The OMX plugin interface provides hooks for vendor-specific OMX core registration and the OMX component plugin.

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

AwesomeRenderer holds the Android native window and handles the video decode output buffer allocation, de-allocation, and rendering to SurfaceFlinger.

NuPlayer works as the player engine to coordinate the above modules, and is connected into the Android media framework through the adapter of the StageFright Player.

2.2 TI MM HAL

TI MM HAL implements OMX Core, OMX components, OSAL for OS primitives (mutex, semaphore, pipes, events), and distributed codec engine (DCE) client.
IVA-HD hardware-accelerated video decoders are integrated into Android as OMX components.

As shown in Figure 5, the core decoder 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, EmptyThisBuffer, 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 decoder and then process the frames.

The OMX framework is designed to split the functionality into logical blocks, OMXBase, Data I/O (DIO), video decoder common, and video decoder-specific modules.

The OMX base implements the base class for the OMX state machine, reused among multiple decoders and encoders. It provides buffer allocation, message passing through pipes, and port handling through DIO.

The OMX video decoder common is derived from the OMXBase, and overrides some of the OMX methods to handle decoder-specific configuration. Those overrides include set/get Config, set/get Param, and so forth.

The OMX VideoDecoder H264 is derived from the OMX video decoder common, and overrides a few methods for handling H264 decoder-specific data structures such as reference frame calculation.

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

2.2.1.1 Buffer Allocation

The codec input buffers are bit-stream buffers, allocated from the DRM/GEM driver directly by the OMX component. Whereas the output buffers for the decoder are meant for display on screen, they are allocated through the GraphicBuffer interface in SurfaceFlinger. TI only supports TILER1D output buffers in an NV-12 format for the hardware decoders. When StageFright, SurfaceFlinger, or Gralloc receives a custom pixel format for NV12 (0x100), PVR allocates the buffers from the GEM system heap. Based on the custom usage flags implemented in the OMX components, OMX components are made aware of the GRALLOC handles for the output buffers. The OMX client deals with only Gralloc handles; the actual buffer FDs are extracted by the OMX component directly from the IMG native handle, and the kernel rpmsg module imports these gem buffer FDs and requests the gem for TILER-1D mapping.

Because the IVA-HD decoder operates on padded output buffers, the actual output buffer dimensions are not known during the component idle state transition. This is handled in StageFright by implementing the port reconfiguration. On processing the first frame, the OMX component sends a portConfiguration changed event and reconfigures the ports with the new dimensions and new set of output buffers.

2.2.1.2 Buffer Flow

Because the decoder modules are integrated as OMX components, the buffer flow follows the OMX standard calls: FillThisBuffer(FTB), EmptyThisBuffer(ETB), FillBufferDone(FBD), and EmptyBufferDone(EBD).

Figure 6. Decoder Buffer Allocation and Dataflow
2.2.1.3 Buffer Synchronization

For the IVA-HD decoder, reference buffers and display buffers are the same. Thus, the decoder output buffers are read-only, and should be synchronized between the IVA-HD and DSS accesses. The GRALLOC interface is used by the OMX component for the output buffer synchronization. This is required because the buffer posting to display is an asynchronous operation. Thus, before returning the buffer to IVA-HD for filling, the OMX component waits on the GRALLOC lock to ensure that the buffer is freed from the display screen.

2.2.2 Audio Video Synchronization

Figure 7. Audio Video Synchronization

Because audio playback is real time at the rate of the sampling frequency, use audio data progress as the reference for the AV sync logic. The playback clock used for the progress bar update is periodically corrected as per the audio data progress to give accurate data to the UI. NuPlayer checks the playback clock, and delays or drops the video frame to be in sync with the audio.

The AVSync algorithm is managed by NuPlayer, which manages the OMXClient as well the AudioPlayer. If the decoded video frame is received with a timestamp more than 1/24 second (40000 ms) late, the video frame is dropped; if the decoded video frame is received in advance of more than 10000 ms, it is kept on hold for the next event, or rendered the frame through NativeWindow.

NuPlayer uses the TimeSource class to get the actual rendering time to take the decision. If no audio track is present, then the TimeSource API is used with the system clock. This is the case with a video-only playback scenario. If an audio track is available, the TimeSource API is implemented by the audioplayer, and the audioplayer provides the audio timestamp (latency data) captured from the audioFlinger/AudioHAL.
3 Interfaces

3.1 HAL Interfaces

3.1.1 OMXBase Structure

```c
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

```c
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,
```
3.1.3 OMXVideoDecoder Structure

typedef struct OMXVideoDecoderComponent {
  OMXBaseComp sBase;
  /* codec related fields */
  OMX_STRING cDecoderName;
  VIDDEC3_Handle pDecHandle;
  Engine_Handle ce;
  IVIDDEC3_Params *pDecStaticParams; /* Pointer to Decoder Static Params */
  IVIDDEC3_DynamicParams *pDecDynParams; /* Pointer to Decoder Dynamic Params */
  IVIDDEC3_Status *pDecStatus; /* Pointer to Decoder Status */
  IVIDDEC3_InArgs *pDecInArgs; /* Pointer to Decoder InArgs */
  IVIDDEC3_OutArgs *pDecOutArgs; /* Pointer to Decoder OutArgs */
  XDM2_BufDesc *tInBufDesc;
  XDM2_BufDesc *tOutBufDesc;

  /* OMX params */
  OMX_VIDEO_PARAM_PORTFORMATTYPE tVideoParams[OMX_VIDDEC_NUM_OF_PORTS];
  OMX_CONFIG_RECTTYPE tCropDimension;
  OMX_CONFIG_SCALEFACTORTYPE tScaleParams;
  OMX_PARAM_COMPONENTROLETYPE tComponentRole;
  OMX_CONFIG_RECTTYPE t2DBufferAllocParams[OMX_VIDDEC_NUM_OF_PORTS];

  /* local params */
  gralloc_module_t const *grallocModule;
  OMXBase_CodecConfigBuf sCodecConfig;
  OMX_U32 nOutPortReconfigRequired;
  OMX_U32 nCodecRecreationRequired;
  OMX_U32 bInputBufferCancelled;
  OMX_U32 bIsFlushRequired;
  OMX_BOOL bUsePortReconfigForCrop;
  OMX_BOOL bUsePortReconfigForPadding;
  OMX_BOOL bSupportDecodeOrderTimeStamp;
  OMX_BOOL bSupportSkipGreyOutputFrames;
  OMX_U32 nFrameCounter;
  OMX_BOOL bSyncFrameReady;
  OMX_U32 nOutbufInUseFlag;
  OMX_PTR pCodecSpecific;
  OMX_U32 nDecoderMode;
  OMX_U32 nFatalErrorGiven;
  OMX_PTR pTimeStampStoragePipe;
  OMX_U32 nFrameRateDivisor;
  OMX_BOOL bFirstFrameHandled;

  void (*fpSet_StaticParams)(OMX_HANDLETYPE hComponent, void *params);
  void (*fpSet_DynamicParams)(OMX_HANDLETYPE hComponent, void *dynamicparams);
  void (*fpSet_Status)(OMX_HANDLETYPE hComponent, void *status);
  void (*fpDeinit_Codec)(OMX_HANDLETYPE hComponent);
  OMX_ERRORTYPE (*fpHandle_ExtendedError)(OMX_HANDLETYPE hComponent);
  OMX_ERRORTYPE (*fpHandle_CodecGetStatus)(OMX_HANDLETYPE hComponent);
  PaddedBuffParams (*fpCalc_OubuffDetails)(OMX_HANDLETYPE hComponent, OMX_U32 width, OMX_U32 height);
} OMX_DIO_Object;
3.1.4 DCE Client Interface (VIDDEC3 Interface)

/*
 * ------- VIDDEC3_control -------
 */
/**
 * @brief Execute the control() method in this instance of a video decoder algorithm.
 *
 * @param[in] handle Handle to a created video decoder instance.
 * @param[in] id Command id for XDM control operation.
 * @param[in] params Runtime control parameters used for decoding.
 * @param[out] status Status info upon completion of decode operation.
 *
 * @pre @c handle is a valid (non-NULL) video decoder handle
 * and the video decoder is in the created state.
 *
 * @retval #VIDDEC3_EOK Success.
 * @retval #VIDDEC3_EFAIL Failure.
 * @retval #VIDDEC3_EUNSUPPORTED Unsupported request.
 *
 * @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 #XDM_ErrorBit for details.
 *
 * @sa VIDDEC3_create()
 * @sa VIDDEC3_delete()
 * @sa IVIDDEC3_Fxns::process()
 */

extern Int32 VIDDEC3_control(VIDDEC3_Handle handle, VIDDEC3_Cmd id,
VIDDEC3_DynamicParams *params, VIDDEC3_Status *status);

/ *
 * ------- VIDDEC3_create -------
 */
/**
 * @brief Create an instance of a video decoder algorithm.
 *
 * Instance handles must not be concurrently accessed by multiple threads;
 * each thread must either obtain its own handle (via VIDDEC3_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 decoder to create.
 * @param[in] params Creation parameters.
 *
 * @retval NULL An error has occurred.
 * @retval non-NULL The handle to the newly created video decoder instance.
 *
 * @remark @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 decoder.
 *
 * @codecNameRemark
 * @sa Engine_open()
extern VIDDEC3_Handle VIDDEC3_create(Engine_Handle e, String name, VIDDEC3_Params *params);

extern Void VIDDEC3_delete(VIDDEC3_Handle handle);

extern Void VIDDEC3_process(VIDDEC3_Handle handle, VIDDEC3_BufferDesc inBufs, VIDDEC3_BufferDesc outBufs, VIDDEC3_Args inArgs, VIDDEC3_Args outArgs);

* @brief Delete the instance of a video decoder algorithm.
* @param[in] handle Handle to a created video decoder instance.
* @remark Depending on the configuration of the engine opened, this call may delete a local or remote instance of the video decoder.
* @pre @c handle is a valid (non-NULL) handle which is in the created state.
* @post All resources allocated as part of the VIDDEC3_create() operation (memory, DMA channels, etc.) are freed.
* @sa VIDDEC3_create()
*/

* @brief Execute the process() method in this instance of a video decoder algorithm.
* @param[in] handle Handle to a created video decoder 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 decoder handle and the video decoder is in the created state.
* @retval #VIDDEC3_EOK Success.
* @retval #VIDDEC3_EFAIL Failure.
* @retval #VIDDEC3_EUNSUPPORTED Unsupported request.
* @remark Since the VIDDEC3 decoder contains support for asynchronous buffer submission and retrieval, this API becomes known as asynchronous in nature.
* @remark This is a blocking call, and will return after the data has been decoded.
* @remark The buffers supplied to VIDDEC3_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 #XDM_ErrorBit for details.
extern Int32 VIDDEC3_process(VIDDEC3_Handle handle, XDM2_BufDesc *inBufs, XDM2_BufDesc *outBufs, VIDDEC3_InArgs *inArgs, VIDDEC3_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 Rc 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
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 xml for Decoder Selection

<MediaCodecs>
  <Decoders>
    <MediaCodec name="OMX.TI.DUCATI1.VIDEO.DECODER">
      <Type name="video/mp4v-es" />
      <Type name="video/3gpp" />
      <Type name="video/avc" />
      <Type name="video/mpeg2" />
      <Quirk name="requires-allocate-on-input-ports" />  
      <Quirk name="requires-allocate-on-output-ports" />
    </MediaCodec>
  </Decoders>
</MediaCodecs>
3.3 MediaPlayer

class MediaPlayer : public BnMediaPlayerClient,
public virtual IMediaDeathNotifier
{
public:
    MediaPlayer();
    ~MediaPlayer();
    void died();
    void disconnect();

    status_t setDataSource(
        const sp<IMediaHTTPService> &httpService,
        const char *url,
        const KeyedVector<String8, String8> *headers);

    status_t setDataSource(int fd, int64_t offset, int64_t length);

    status_t setVideoSurfaceTexture(
        const sp<IGraphicBufferProducer>& bufferProducer);

    status_t setListener(const sp<MediaPlayerListener>& listener);

    status_t prepare();
    status_t prepareAsync();
    status_t start();
    status_t stop();
    status_t pause();

    bool isPlaying();

    status_t getVideoWidth(int *w);
    status_t getVideoHeight(int *h);
    status_t seekTo(int64_t msec);
    status_t getDuration(int *msec);
    status_t reset();

    status_t setAudioStreamType(audio_stream_type_t type);
    status_t getAudioStreamType(audio_stream_type_t *type);

    status_t invoke(const Parcel& request, Parcel *reply);
    status_t setMetadataFilter(const Parcel& filter);

    status_t setAuxEffectSendLevel(float level);
    status_t attachAuxEffect(int effectId);

    status_t setParameter(int key, const Parcel& request);
    status_t getParameter(int key, Parcel* reply);

    status_t setRetransmitEndpoint(const char* addrString, uint16_t port);

private:
    void clear_l();
    status_t seekTo_l(int msec);
    status_t prepareAsync_l();
    status_t getDuration_l(int *msec);
status_t attachNewPlayer(const sp<IMediaPlayer>& player);
status_t reset_l();
status_t doSetRetransmitEndpoint(const sp<IMediaPlayer>& player);
status_t checkStateForKeySet_l(int key);

sp<IMediaPlayer> mPlayer;
thread_id_t mLockThreadId;
Mutex mLock;
Mutex mNotifyLock;
Condition mSignal;
sp<MediaPlayerListener> mListener;
void* mCookie;
media_player_states mCurrentState;
int mCurrentPosition;
int mSeekPosition;
bool mPrepareSync;
status_t mPrepareStatus;
audio_stream_type_t mStreamType;
Parcel* mAudioAttributesParcel;
bool mLoop;
float mLeftVolume;
float mRightVolume;
int mVideoWidth;
int mVideoHeight;
int mAudioSessionId;
float mSendLevel;
struct sockaddr_in mRetransmitEndpoint;
bool mRetransmitEndpointValid;
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

<table>
<thead>
<tr>
<th>Audio</th>
<th>Amplifiers</th>
<th>Data Converters</th>
<th>DLP® Products</th>
<th>DSP</th>
<th>Clocks and Timers</th>
<th>Interface</th>
<th>Logic</th>
<th>Power Mgmt</th>
<th>Microcontrollers</th>
<th>RFID</th>
<th>OMAP Applications Processors</th>
<th>Wireless Connectivity</th>
</tr>
</thead>
</table>

Applications

<table>
<thead>
<tr>
<th>Automotive and Transportation</th>
<th>Communications and Telecom</th>
<th>Computers and Peripherals</th>
<th>Consumer Electronics</th>
<th>Energy and Lighting</th>
<th>Industrial</th>
<th>Medical</th>
<th>Security</th>
<th>Space, Avionics and Defense</th>
<th>Video and Imaging</th>
<th>TI E2E Community</th>
<th>e2e.ti.com</th>
</tr>
</thead>
</table>

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

Copyright © 2016, Texas Instruments Incorporated