# MPEG2 Main Profile Decoder on HDVICP2 and Media Controller Based Platform

## User's Guide


Literature Number: SPRUH45
February 2012

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI 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 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

| **Products** | | **Applications** | |
|---|---|---|---|
| Amplifiers | amplifier.ti.com | Audio | www.ti.com/audio |
| Data Converters | dataconverter.ti.com | Automotive | www.ti.com/automotive |
| DLP® Products | www.dlp.com | Broadband | www.ti.com/broadband |
| DSP | dsp.ti.com | Digital Control | www.ti.com/digitalcontrol |
| Clocks and Timers | www.ti.com/clocks | Medical | www.ti.com/medical |
| Interface | interface.ti.com | Military | www.ti.com/military |
| Logic | logic.ti.com | Optical Networking | www.ti.com/opticalnetwork |
| Power Mgmt | power.ti.com | Security | www.ti.com/security |
| Microcontrollers | microcontroller.ti.com | Telephony | www.ti.com/telephony |
| RFID | www.ti-rfid.com | Video & Imaging | www.ti.com/video |
| RF/IF and ZigBee® Solutions | www.ti.com/lprf | Wireless | www.ti.com/wireless |

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

# Preface

# Read This First

## *About This Manual*

This document describes how to install and work with Texas Instruments' (TI) MPEG2 Main Profile Decoder implementation on the HDVICP2 and Media Contrller based platform. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

TI's codec implementations are based on the eXpressDSP Digital Media (XDM) standard. XDM is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS).

## *Intended Audience*

This document is intended for system engineers who want to integrate TI's codecs with other software to build a multimedia system based on the HDVICP2 based platform.

This document assumes that you are fluent in the C language, have a good working knowledge of Digital Signal Processing (DSP), digital signal processors, and DSP applications. Good knowledge of eXpressDSP Algorithm Interface Standard (XDAIS) and eXpressDSP Digital Media (XDM) standard will be helpful.

## *How to Use This Manual*

This document includes the following chapters:

- **Chapter 1 - Introduction**, provides a brief introduction to the XDAIS and XDM standards. It also provides an overview of the codec and lists its supported features.
- **Chapter 2 - Installation Overview**, describes how to install, build, and run the codec.
- **Chapter 3 - Sample Usage**, describes the sample usage of the codec.
- **Chapter 4 - API Reference**, describes the data structures and interface functions used in the codec.
- **Chapter 5 – Frequently Asked Questions,** answers few frequently asked questions related to using MPEG2 Main Profile Decoder on HDVICP2 and Media Controller Based Platform.
- **Chapter 6 – Picture Format,** provides information on format of YUV buffers provided to decoder.

- **Chapter 7 – Debug Trace Usage,** describes the debug trace feature supported by the codec and its usage.
- **Chapter 8 – Error Handling,** explains the error handling and error robustness features of this MPEG-2 Decoder.

### *Related Documentation From Texas Instruments*

The following documents describe TI's DSP algorithm standards such as, XDAIS and XDM. To obtain a copy of any of these TI documents, visit the Texas Instruments website at www.ti.com.

- *TMS320 DSP Algorithm Standard Rules and Guidelines* (literature number SPRU352) defines a set of requirements for DSP algorithms that, if followed, allow system integrators to quickly assemble production-quality systems from one or more such algorithms.
- *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360) describes all the APIs that are defined by the TMS320 DSP Algorithm Inteface Standard (also known as XDAIS) specification.
- *Technical Overview of eXpressDSP - Compliant Algorithms for DSP Software Producers* (literature number SPRA579) describes how to make algorithms compliant with the TMS320 DSP Algorithm Standard which is part of TI's eXpressDSP technology initiative.
- *Using the TMS320 DSP Algorithm Standard in a Static DSP System* (literature number SPRA577) describes how an eXpressDSP-compliant algorithm may be used effectively in a static system with limited memory.
- eXpressDSP Digital Media (XDM) Standard API Reference (literature number SPRUEC8)
- *Using IRES and RMAN Framework Components for C64x+* (literature number SPRAAI5), describes the IRES interface definition and function calling sequence

### *Related Documentation*

You can use the following documents to supplement this user guide:

- ISO/IEC-13838-2: Information Technology- Generic coding of moving pictures and associated audio information: Video

### *Abbreviations*

The following abbreviations are used in this document.

*Table 0-1. List of Abbreviations*

| Abbreviation | Description |
|---|---|
| BIOS | TI's simple RTOS for DSPs |

| Abbreviation | Description |
|---|---|
| CSL | Chip Support Library |
| D1 | 720x480 or 720x576 resolutions in progressive scan |
| DCT | Discrete Cosine Transform |
| DMA | Direct Memory Access |
| DMAN | DMA Manager |
| DPB | Decoded Picture Buffer |
| EVM | Evaluation Module |
| HDTV | High Definition Television |
| IRES | Interface standard to request and receive handles to resources |
| ISO | International Standards Organization |
| HDVICP2 | Image Video Accelerator |
| MB | Macro Block |
| MPEG | Moving Pictures Experts Group |
| MV | Motion Vector |
| NAL | Network Adaptation Layer |
| NTSC | National Television Standards Committee |
| RMAN | Resource Manager |
| RTOS | Real Time Operating System |
| VCL | Video Coding Layer |
| VGA | Video Graphics Array (640 x 480 resolution) |
| VOP | Video Object Plane |
| XDAIS | eXpressDSP Algorithm Interface Standard |
| XDM | eXpressDSP Digital Media |
| YUV | Color space in luminance and chrominance form |

## Text Conventions

The following conventions are used in this document:

- Text inside back-quotes (‘‘) represents pseudo-code.
- Program source code, function and macro names, parameters, and command line commands are shown in a `mono-spaced` font.

## Product Support

When contacting TI for support on this codec, quote the product name (MPEG2 Main Profile Decoder on HDVICP2) and version number. The version number of the codec is included in the title of the Release Notes that accompanies this codec.

## Trademarks

Code Composer Studio, DSP/BIOS, eXpressDSP, TMS320, HDVICP2,are trademarks of Texas Instruments.

All trademarks are the property of their respective owners.

# This page is intentionally left blank

# Contents

# Figures

# This page is intentionally left blank

# Tables

Chapter 1

# Introduction

This chapter provides a brief introduction to XDAIS and XDM. It also provides an overview of TI's implementation of the MPEG2 Main Profile Decoder on the HDVICP2 and Media Controller based platform and its supported features.

**Topic** **Page**

## 1.1 Overview of XDAIS and XDM

TI's multimedia codec implementations are based on the eXpressDSP Digital Media (XDM) standard. XDM is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS).

### *1.1.1 XDAIS Overview*

An eXpressDSP-compliant algorithm is a module that implements the abstract interface IALG. The IALG API takes the memory management function away from the algorithm and places it in the hosting framework. Thus, an interaction occurs between the algorithm and the framework. This interaction allows the client application to allocate memory for the algorithm and also share memory between algorithms. It also allows the memory to be moved around while an algorithm is operating in the system. In order to facilitate these functionalities, the IALG interface defines the following APIs:

- `algAlloc()`
- `algInit()`
- `algActivate()`
- `algDeactivate()`
- `algFree()`

The `algAlloc()` API allows the algorithm to communicate its memory requirements to the client application. The `algInit()` API allows the algorithm to initialize the memory allocated by the client application. The `algFree()` API allows the algorithm to communicate the memory to be freed when an instance is no longer required.

Once an algorithm instance object is created, it can be used to process data in real-time. The `algActivate()` API provides a notification to the algorithm instance that one or more algorithm processing methods is about to be run zero or more times in succession. After the processing methods have been run, the client application calls the `algDeactivate()` API prior to reusing any of the instance's scratch memory.

The IALG interface also defines three more optional APIs `algControl()`, `algNumAlloc()`, and `algMoved()`. For more details on these APIs, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

### 1.1.2 XDM Overview

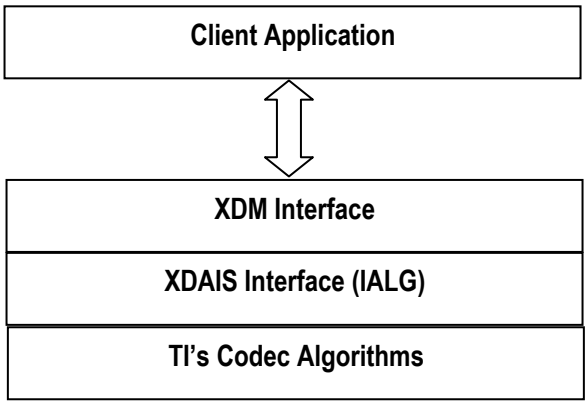
In the multimedia application space, you have the choice of integrating any codec into your multimedia system. For example, if you are building a video decoder system, you can use any of the available video decoders (such as MPEG4, H.263, or MPEG-2) in your system. To enable easy integration with the client application, it is important that all codecs with similar functionality use similar APIs. XDM was primarily defined as an extension to XDAIS to ensure uniformity across different classes of codecs (for example audio, video, image, and speech). The XDM standard defines the following two APIs:

- `control()`
- `process()`

The `control()` API provides a standard way to control an algorithm instance and receive status information from the algorithm in real-time. The `control()` API replaces the `algControl()` API defined as part of the IALG interface. The `process()` API does the basic processing (encode/decode) of data.

Apart from defining standardized APIs for multimedia codecs, XDM also standardizes the generic parameters that the client application must pass to these APIs. The client application can define additional implementation specific parameters using extended data structures.

The following figure depicts the XDM interface to the client application.




As depicted in the figure, XDM is an extension to XDAIS and forms an interface between the client application and the codec component. XDM insulates the client application from component-level changes. Since TI's multimedia algorithms are XDM compliant, it provides you with the flexibility to use any TI algorithm without changing the client application code. For example, if you have developed a client application using an XDM-compliant MPEG4 video decoder, then you can easily replace MPEG4 with another XDM-compliant video decoder, say H.263, with minimal changes to the client application.

For more details, see *eXpressDSP Digital Media (XDM) Standard API Reference* (literature number SPRUEC8).

### 1.1.3 IRES Overview

IRES is a generic, resource-agnostic, extendible resource query, initialization and activation interface. The application framework defines, implements, and supports concrete resource interfaces in the form of IRES extensions. Each algorithm implements the generic IRES interface, to request one or more concrete IRES resources. IRES defines standard interface functions that the framework uses to query, initialize, activate/deactivate and reallocate concrete IRES resources. To create an algorithm instance within an application framework, the algorithm and the application framework agrees on the concrete IRES resource types that are requested. The framework calls the IRES interface functions, in addition to the IALG functions, to perform IRES resource initialization, activation, and deactivation.

The IRES interface introduces support for a new standard protocol for cooperative preemption, in addition to the IALG-style non-cooperative sharing of scratch resources. Co-operative preemption allows activated algorithms to yield to higher priority tasks sharing common scratch resources. Framework components include the following modules and interfaces to support algorithms requesting IRES-based resources:

- **IRES** - Standard interface allowing the client application to query and provide the algorithm with its requested IRES resources.
- **RMAN** - Generic IRES-based resource manager, which manages and grants concrete IRES resources to algorithms and applications. RMAN uses a new standard interface, the IRESMAN, to support run-time registration of concrete IRES resource managers.

Client applications call the algorithm's IRES interface functions to query its concrete IRES resource requirements. If the requested IRES resource type matches a concrete IRES resource interface supported by the application framework, and if the resource is available, the client grants the algorithm logical IRES resource handles representing the allotted resources. Each handle provides the algorithm with access to the resource as defined by the concrete IRES resource interface.

IRES interface definition and function calling sequence is depicted in the following figure. For more details, see *Using IRES and RMAN Framework Components for C64x+* (literature number SPRAAI5).

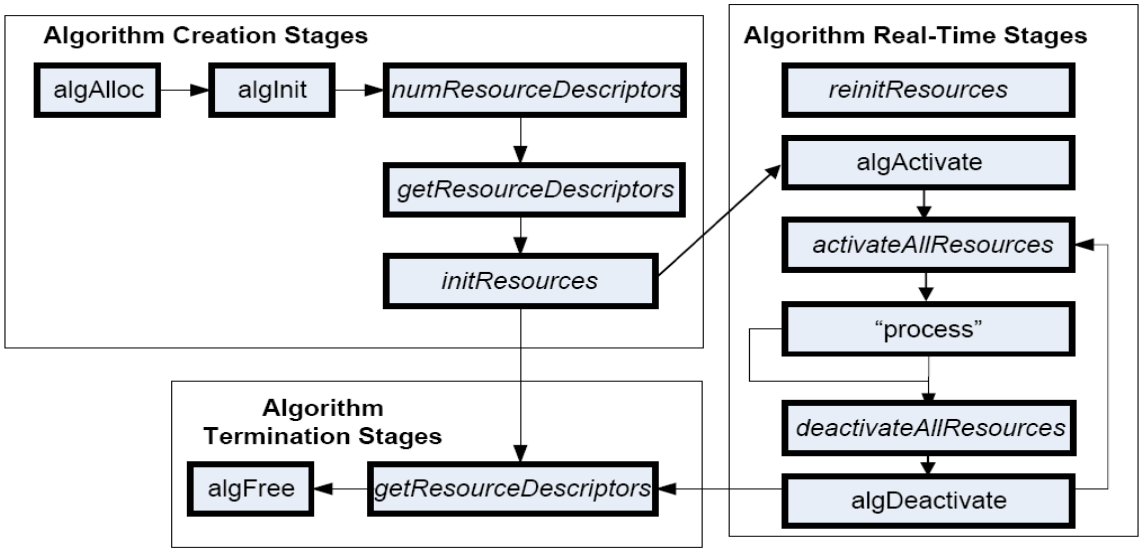



*Figure 1-1. IRES Interface Definition and Function Calling Sequence*

For more details, see *Using IRES and RMAN Framework Components for C64x+* (literature number SPRAAI5).

## 1.2 Overview of MPEG2 Main Profile Decoder

MPEG-2 is a widely used video compression algorithm that uses motion compensated prediction, Discrete Cosine Transform (DCT) coding of the prediction error signal and modified Huffman entropy coding.

Please refer to ISO/IEC-13838-2 for detailed algorithm and working of MPEG-2 decoder.

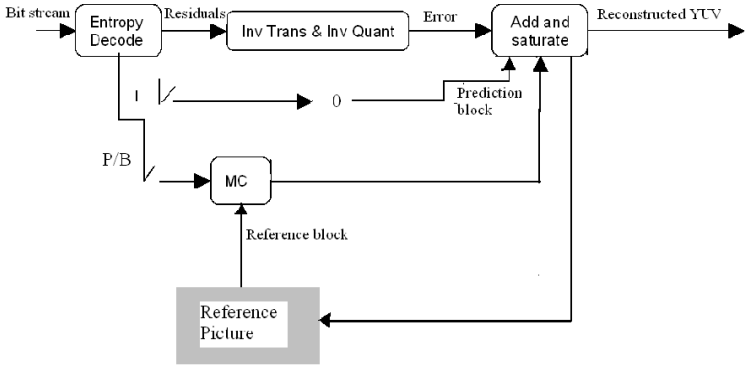



*Figure 1-2. Flow Diagram of the MPEG2 Decoder*

From this point onwards, all references to MPEG2 Decoder means MPEG2 Main Profile Decoder only.

## 1.3 Supported Services and Features

This user guide accompanies TI's implementation of MPEG2 Decoder on the HDVICP2 based platform.

This version of the codec has the following supported features:

- eXpressDSP Digital Media (XDM IVIDDEC3) compliant
- Uses hardware accelerators of HDVICP2
- Supports up to High level of Main Profile (MP)
- Supports decoding of MPEG-1 constrained video streams
- Supports progressive and interlaced type picture decoding
- Supports P and B frames
- Supports frame based decoding
- Supports picture width and height (resolutions) greater than 64 pixels including all standard resolutions up to 2048x2048
- Supports optional out-of-loop deblocking for display
- Supports graceful exit under error conditions
- Supports YUV420 semi-planar chroma format
- Independent of any Operating System
- Has logic and checks to help application to support trick mode playback
- Supports multi-channel functionality
- Supports dump of macroblock information (useful in transcode applications)
- Supports error concealment
- Supports debug trace dump

Chapter 2

# Installation Overview

This chapter provides a brief description on the system requirements and instructions for installing the codec component. It also provides information on building and running the sample test application.

**Topic** **Page**

## 2.1 System Requirements

This section describes the hardware and software requirements for the normal functioning of the codec component.

### *2.1.1 Hardware*

This codec has been tested on the HDVICP2 and Media Controller based OMAP4 ES1.0 and DM816x DDR2 EVM REV-B hardware platforms.

### *2.1.2 Software*

The following are the software requirements for the normal functioning of the codec:

- **Development Environment:** This project is developed using Code Composer Studio (Code Composer Studio v4) version 4.2.0.09000.

  http://software-dl.ti.com/dsps/dsps_registered_sw/sdo_ccstudio/CCSv4/Prereleases/setup_CCS_4.2.0.09000.zip

- **Code Generation Tools:** This project is compiled, assembled, archived, and linked using the code generation tools version 4.5.1.

  Although CG tools v 4.5.1 is a part of Code Composer Studio v4 installation, it is recommended that you re-install CG tools by downloading from the following link.

  https://www-a.ti.com/downloads/sds_support/CodeGenerationTools.htm

- **HDVICP2 Simulator:** This codec has been tested using HDVICP2 Simulator version 5.0.16 (HDVICP2 Simulation CSP 1.1.5). This release can be obtained by software updates on Code Composer Studio v4. Ensure that the following site is listed as part of “Update sites to visit”.

  http://software-dl.ti.com/dsps/dsps_public_sw/sdo_ccstudio/CCSv4/Updates/IVAHD/site.xml

  This codec has also been tested using Netra CSP (Simulation) version 0.7.1. This version of Simulator can be downloaded through software updates on Code Composer Studio v4. Ensure that the following site is listed as part of “Update sites to visit”.

  http://software-dl.ti.com/dsps/dsps_public_sw/sdo_ccstudio/CCSv4/Updates/NETRA/site.xml

## 2.2 Installing the Component

The codec component is released as a compressed archive. To install the codec, extract the contents of the zip file onto your local hard disk. The zip file extraction creates a top-level directory called 500.V.MPEG2.D.IVAHD.01.00 under which under which the directory named IVAHD_001 is created:

Figure 2-1 shows the sub-directories created in the IVAHD_001 directory.


*Figure 2-1. Component Directory Structure*

Table 2-1 provides a description of the sub-directories created in the 500.V.MPEG2.D.IVAHD.01.00 directory.

| **Sub-Directory** | **Description** |
|---|---|
| \client\build\TestAppDeviceName | Contains the Media Controller cmd file. The name of this directory will not be same as exactly mentioned here. Instead of DeviceName string, actual name of Device will be present. |
| \client\build\TestAppDeviceName\make | Contains the make file for the test application project. The name of this directory will not be same as exactly mentioned here. Instead of DeviceName string, actual name of Device will be present. |
| \client\build\TestAppDeviceName\map | Contains the memory map generated on compilation of the code |
| \client\build\TestAppDeviceName\obj | Contains the intermediate .asm and/or .obj file generated on compilation of the code |
| \client\build\TestAppDeviceName\Out | Contains the final application executable (.out) file generated by the sample test application |
| \client\test\inc | Contains header files needed for the application code |
| \client\test\src | Contains application C files |

| Sub-Directory | Description |
|---|---|
| \client\test\testvecs\config | Contains sample configuration file for MPEG-2 Decoder |
| \client\test\testvecs\input | Contains input test vectors |
| \client\test\testvecs\output | Contains output generated by the codec. It is empty directory as part of release. |
| \client\test\testvecs\reference | Contains read-only reference output to be used for cross-checking against codec output |
| \docs | Contains user guide, data sheet |
| \inc | Contains interface header files of MPEG-2 Decoder |
| \lib | Contains jpegvdec_ti_host.lib – HDVICP2 MPEG-2 Decoder built as a library on Media Controller |

*Table 2-1. Component Directories*

## 2.3 Before Building the Sample Test Application

This codec is accompanied by a sample test application. To run the sample test application, you need TI Framework Components (FC).

This version of the codec has been validated with Framework Components (FC) version 3.20.00.22 GA.

To run the Simulator version of the codec, the HDVICP2 simulator has to be installed. The version of the simulator is 5.0.16. This can be done using the "Help->Software Updates->Find and Install" option in CCSv4. Detailed instructions to set up the configuration can be found in ivahd_sim_user_guide.pdf present in <CCSv4 Installation Dir>\simulation_csp_omap4\docs\pdf\ directory.

This codec has also been validated on Netra Video Processing Simulator that simulates all the three HDVICP2s in DM816x. The simulator required for this is Netra CSP (Simulation) version 0.7.1. This simulator can also be installed using the "Help->Software Updates->Find and Install" option in CCSv4. Detailed instructions to set up the configuration can be found in netra_sim_user_guide.pdf present in <CCSv4 Installation Dir>\ simulation_netra\docs\user_guide directory.

Install CG Tools version 4.5.1 for ARM (TMS470) at the following location in your system: <CCSv4.2_InstallFolder>\ccsv4\tools\compiler\tms470.

CGTools 4.5.1 can be downloaded from

https://www-a.ti.com/downloads/sds_support/CodeGenerationTools.htm

Please note that CG Tools 4.5.1 is installed at the location mentioned above along with the CCS v4.2 installation by default. But, as some problems have been reported about this, we recommend that you install CG Tools 4.5.1 again with the installer obtained from the above link.

Set environment variable CG_TOOL_DIR to <cgtools_install_dir>.

<CG_TOOL_DIR>/bin should contain all required code generation tools executables.

Set environment variables HDVICP2_INSTALL_DIR and CSP_INSTALL_DIR to the locations where the HDVICP20 API library and HDVICP2 CSL are present. The HDVICP20 API library and the HDVICP2 CSL can be downloaded from the same place as the codec package. The HDVICP20 API .lib files should be present at HDVICP2_INSTALL_DIR/lib and HDVICP20 API interface header files at HDVICP2_INSTALL_DIR/inc. The folders csl_ivahd and csl_soc of HDVICP2 CSL should be present at CSP_INSTALL_DIR/.

This version of the codec has been validated with HDVICP2.0 API library version 01.00.00.19 and HDVICP2.0 CSL Version 00.05.02.

Set the system environment variable TI_DIR to the CCSv4 installation path. Example: TI_DIR = <CCSv4 Installation Dir>\ccsv4.

Add gmake (GNU Make version 3.78.1) utility folder path (for example, “C:\CCStudioV4.0\ccsv4\utils\gmake”) at the beginning of the PATH environment variable.

The version of the XDC tools required is 3.20.04.68 GA.

### 2.3.1 Installing Framework Component (FC)

You can download FC from the TI website:

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/fc/3_20_00_22/index_FDS.html

Extract the FC zip file to the some location and set the system environment variable FC_INSTALL_DIR to this path. For example: if the zip file was extracted to C:\CCSv4\, set FC_INSTALL_DIR as C:\CCSv4\framework_components_3_20_00_22.

The test application uses the following IRES and XDM files:

- HDVICP related IRES header files, these are available in the FC_INSTALL_DIR\packages\ti\sdo\fc\ires\hdvicp directory.
- Tiled memory related Header file, these are available in the FC_INSTALL_DIR\fctools\packages\ti\sdo\fc\ires\tiledmemory directory.
- XDM related header files, these are available in the FC_INSTALL_DIR\fctools\packages\ti\xdais directory

### 2.3.2 Installing XDC Tools

XDC Tools is required to build the test application. The test application uses the standard files like <std.h> from XDC tools. The xdc tools can be downloaded and installed from the following URL:

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/rtsc/3_20_04_68/index_FDS.html

Also, ensure that the environment variable XDCROOT is set to the XDC installation directory.

## 2.4 Building and Running the Sample Test Application

### *2.4.1 Building the Sample Test Application*

MPEG-2 decoder on HDVICP2 and Media Controller based platform has the following projects.

| Project | Make file Path | Output files |
|---|---|---|
| Test Application | \client\build\<TestAppDeviceName>\make\ | \client\build\TestApp<DeviceName>\out\mpeg2vdec_ti_hosttestapp.out |

The make file for the project can be built using the following commands.

```
gmake -k -s deps

gmake -k -s all
```

Use the following command to clean previous builds.

```
gmake -k -s clean
```

### *2.4.2 Running the Sample Test Application on Netra HDVICP2 Simulator*

The sample test application that accompanies this codec component will run in TI's Code Composer Studio development environment. To run the sample test application on HDVICP2 Simulator, follow these steps:

1) Ensure that you have installed IVAHD CSP (Simulation) version 1.1.5.

2) Start Code Composer Studio v4 and set up the target configuration for Netra IVA-HD Simulator.

3) Select the Debug perspective in the workbench. Launch Netra IVA-HD simulator in CCSv4 (**View > Target Configurations > %Netra Simulator%**).

4) Select M3_Video device and **Target > Load Program**, browse to the \500.V.MPEG2.D.IVAHD.01.00\IVAHD_001\client\build\TestAppDeviceName\out\ sub-directory, select the codec executable "mpeg2vdec_ti_hosttestapp.out" and load it into Code Composer Studio in preparation for execution.

5) Select HDVICP2_0_ICONT1 device and **Target > Run** to give iCont1 device a free run.

6) Select HDVICP2_0_ICONT2 device and **Target > Run** to give iCont2 device a free run.

7) Select **Target > Run** to execute the application for M3_Video device.

8) Test application will take input streams from \500.V.MPEG2.D.IVAHD.01.00\IVAHD_001\client\test\testvecs\input\ directory and generates outputs in \500.V.MPEG2.D.IVAHD.01.00\IVAHD_001\client\test\testvecs\output\ directory.

### *2.4.3 Running the Sample Test Application on DM816x EVM*

The sample test application that accompanies this codec component will run in TI's Code Composer Studio development environment. To run the sample test application on DM816x EVM, follow these steps:

1) Start Code Composer Studio v4 and set up the target configuration for DM816x EVM Emulator.

2) Ensure that the clock is enabled for Media Controller and HDVICP2.

3) Select the Debug perspective in the workbench. Launch DM816x EVM Emulator in CCSv4 (**View > Target Configurations > %DM816x EVM%**).

4) Select Cortex_M3_RTOS_0 device, right click and choose "Connect Target" and wait for emulator to connect to Media Controller

5) Select Cortex_M3_RTOS_0 device and **Target > Load Program**, browse to \500.V.MPEG2.D.IVAHD.01.00\ IVAHD_001\client\build\TestAppDM816x\out\ sub-directory, select the codec executable "mpeg2vdec_ti_hosttestapp.out" and load it in preparation for execution.

6) Select **Target > Run** to execute the application for Cortex_M3_RTOS_0 device.

7) Test application will take input streams from \500.V.MPEG2.D.IVAHD.01.00\IVAHD_001\client\test\testvecs\input\ directory and generates outputs in \500.V.MPEG2.D.IVAHD.01.00\IVAHD_001\client\test\testvecs\output\ directory.

**Note:**

Order of connecting to the devices is important and it should be as mentioned in above steps.

## 2.5 Configuration Files

This codec is shipped along with:

- Generic configuration file (Testvecs.cfg) – specifies input and reference files for the sample test application.
- Decoder configuration file (Testparams.cfg) – specifies the configuration parameters used by the test application to configure the Decoder.

### *2.5.1 Generic Configuration File*

The sample test application shipped along with the codec uses the configuration file, Testvecs.cfg for determining the input and reference files for running the codec and checking for compliance. The Testvecs.cfg file is available in the \Client\Test\TestVecs\Config sub-directory.

The format of the Testvecs.cfg file is:

```
X
Config
Input
Output or Reference
SizeFile
```

where:

- `X` may be set as:
  - 1 - for compliance checking.
  - 0 - for writing the output to the output file
- `Config` is the Decoder configuration file. For details, see Section 2.5.2
- `Input` is the input file name (use complete path).
- `Output` is the output .yuv file name (output dump mode). `Reference` is the reference .yuv file name (in compliance checking mode).
- `SizeFile` is the file containing offsets and size of each frame for TST_FRAME_MODE.

A sample Testvecs.cfg file is as shown:

```
0
..\..\Test\TestVecs\Config\Testparams.cfg
..\..\Test\TestVecs\Input\davincieffect_qcif_25fps.mp2
..\..\Test\TestVecs\Output\davincieffect_qcif_25fps.yuv
..\..\Test\TestVecs\Config\sizeFile.cfg
```

In compliance mode of operation, the decoder compares the reference and the generated output and declares Pass/Fail message. If output dump mode is selected(X set to 0), then the decoder dumps the output to the specified file.

A sample SizeFile is as shown:

```
/* The first field gives offset from
start of input file for each frame */

/* The second field gives frame size
+3, i.e size including the start code
of next picture */

0     4840

4837  4475

9309  978

10284 958

11239 6147

/* To indicate no more input frames */

0     0
```

Note that an additional 3 bytes should be added to the frame size (to include the next start code).

### *2.5.2 Decoder Configuration File*

The decoder configuration file, Testparams.cfg contains the configuration parameters required for the decoder. The Testparams.cfg file is available in the \Client\Test\TestVecs\Config sub-directory.

A sample Testparams.cfg file is as shown:

```
# New Input File Format is as follows
# <ParameterName> = <ParameterValue> # Comment

##########################################################
# Parameters
##########################################################
ImageWidth              = 2048    # Image width in Pels
ImageHeight             = 2048    # Image height in Pels
ChromaFormat            = 9   # 256 =>XDM_YUV_420SP
FramesToDecode          = 100 # Number of frames to be coded
DumpFrom                = 0  # Start dumping from this frame
```

**Note:**

ChromaFormat supported in this codec is 420 semi-planar, that is, the chroma planes (Cb and Cr) are interleaved.

## 2.6 Uninstalling the Component

To uninstall the component, delete the codec directory from your hard disk.

# This page is intentionally left blank

Chapter 3

# Sample Usage

This chapter provides a detailed description of the sample test application that accompanies this codec component.

| Topic | Page |
|---|---|

## 3.1 Overview of the Test Application

The test application exercises the `IVIDDEC3` base class of the MPEG2 Decoder library. The main test application files are mpeg2vdec_ti_hosttestapp.c and mpeg2vdec_ti_rmanConfig.c. These files are available in the \dec\mpeg2\client\test\src directory.

Figure 3-1 depicts the sequence of APIs exercised in the sample test application. Currently, the test application does not use RMAN resource manager. However, all the resource allocations happens through IRES interfaces.

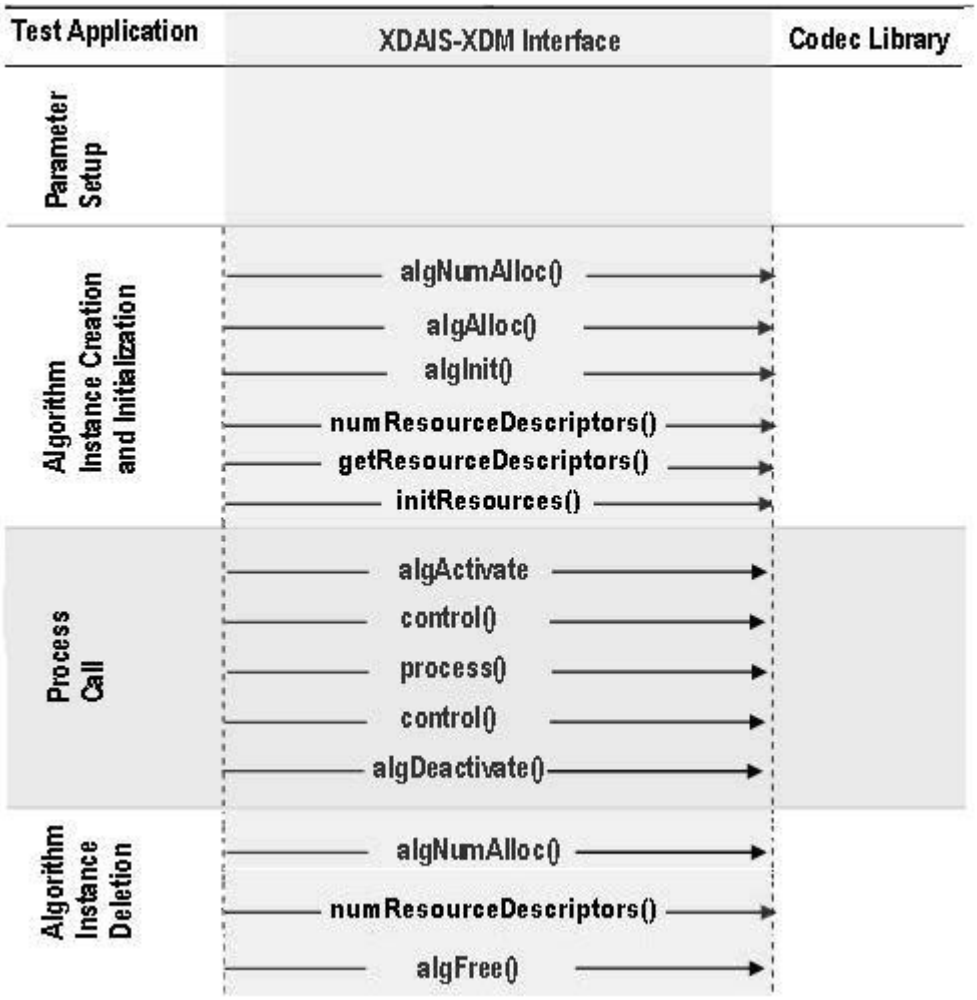



*Figure 3-1. Test Application Sample Implementation*

The test application is divided into four logical blocks:

- Parameter setup
- Algorithm instance creation and initialization
- Process call
- Algorithm instance deletion

### 3.1.1 Parameter Setup

Each codec component requires various codec configuration parameters to be set at initialization. For example, a video codec requires parameters such as video height, video width, and so on. The test application obtains the required parameters from the Decoder configuration files.

In this logical block, the test application does the following:

Opens the generic configuration file, Testvecs.cfg and reads the compliance checking parameter, Decoder configuration file name (Testparams.cfg), input file name, and output/reference file name.

Opens the Decoder configuration file, (Testparams.cfg) and reads the various configuration parameters required for the algorithm. For more details on the configuration files, see Section 0.

Sets the `IVIDDEC3_Params` structure based on the values it reads from the Testparams.cfg file.

Reads the input bit-stream into the application input buffer.

After successful completion of these steps, the test application does the algorithm instance creation and initialization.

### 3.1.2 Algorithm Instance Creation and Initialization

In this logical block, the test application accepts the various initialization parameters and returns an algorithm instance pointer. The following APIs are called in sequence:

`algNumAlloc()` - To query the algorithm about the number of memory records it requires.

`algAlloc()` - To query the algorithm about the memory requirement to be filled in the memory records.

`algInit()` - To initialize the algorithm with the memory structures provided by the application.

A sample implementation of the create function that calls `algNumAlloc()`, `algAlloc()`, and `algInit()` in sequence is provided in the `ALG_create()` function implemented in the alg_create.c file.

**Note:**

- Decoder requests only one memory buffer through `algNumAlloc`. This buffer is for the algorithm handle.
- Other memory buffer requirements are done through IRES interfaces.

After successful creation of the algorithm instance, the test application does HDVICP Resource and memory buffer allocation for the algorithm. Currently, RMAN resource manager is not used. However, all the resource allocations happen through IRES interfaces:

`numResourceDescriptors()` - To understand the number of resources (HDVICP and buffers) needed by algorithm.

`getResourceDescriptors()` – To get the attributes of the resources.

`initResources()` - After resources are created, application gives the resources to algorithm through this API.

### *3.1.3 Process Call*

After algorithm instance creation and initialization, the test application does the following:

Sets the dynamic parameters (if they change during run-time) by calling the `control()` function with the `XDM_SETPARAMS` command.

Sets the input and output buffer descriptors required for the `process()` function call. The input and output buffer descriptors are obtained by calling the `control()` function with the `XDM_GETBUFINFO` command.

Implements the process call based on the non-blocking mode of operation explained in step 4. The behavior of the algorithm can be controlled using various dynamic parameters (see Section 4.2.1.8). The inputs to the `process()` functions are input and output buffer descriptors, pointer to the `IVIDDEC3_InArgs` and `IVIDDEC3_OutArgs` structures.

On the call to the `process()` function for encoding/decoding a single frame of data, the software triggers the start of encode/decode. After triggering the start of the encode/decode frame, the video task can be put to `SEM-pend` state using semaphores. On receipt of interrupt signal at the end of frame encode/decode, the application releases the semaphore and resume the video task, which does any book-keeping operations by the codec and updates the output parameter of `IVIDDEC3_OutArgs` structure.

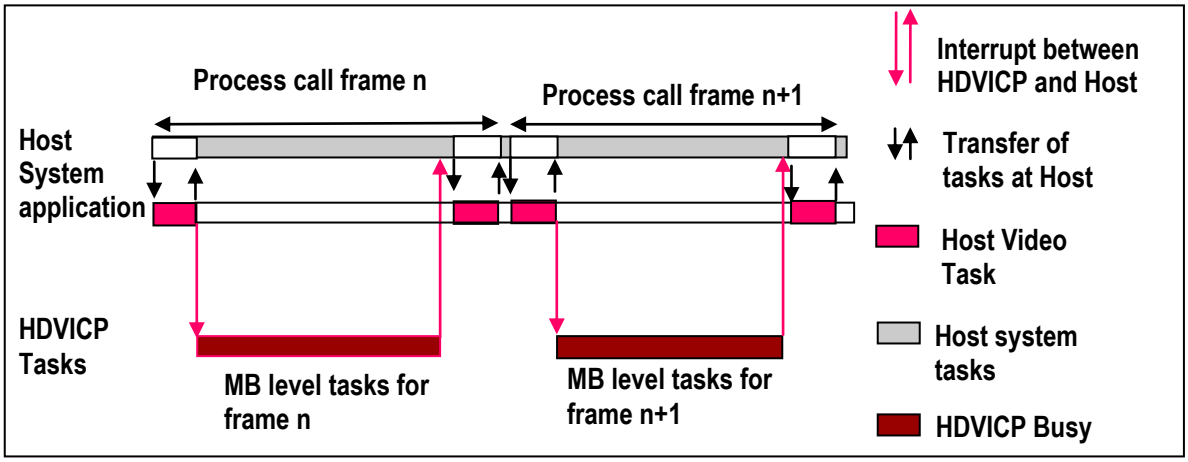



*Figure 3-2. Process call with Host release*

The `control()` and `process()` functions should be called only within the scope of the `algActivate()` and `algDeactivate()` XDAIS functions which activate and deactivate the algorithm instance respectively. Once an algorithm is activated, there could be any ordering of `control()` and `process()` functions. The following APIs are called in a sequence:

`algActivate()` - To activate the algorithm instance.

`control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the six available control commands.

`process()` - To call the Decoder with appropriate input/output buffer and arguments information.

`control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the six available control commands.

`algDeactivate()` - To deactivate the algorithm instance.

The do-while loop encapsulates picture level `process()` call and updates the input buffer pointer every time before the next call. The do-while loop breaks off either when an error condition occurs or when the input buffer exhausts. It also protects the `process()` call from file operations by placing appropriate calls for cache operations. The test application does a cache invalidate for the valid input buffers before `process()` and a cache write back invalidate for output buffers after a `control()` call with `GET_STATUS` command.

In the sample test application, after calling `algDeactivate()`, the output data is either dumped to a file or compared with a reference file.

### 3.1.4 Algorithm Instance Deletion

Once decoding/encoding is complete, the test application frees the memory resources and deletes the current algorithm instance. The following APIs are called in sequence:

`numResourceDescriptors()` - To get the number of resources and free them. If the application needs handles to the resources, it can call `getResourceDescriptors()`.

`algNumAlloc()` - To query the algorithm about the number of memory records it used.

`algFree()` - To query the algorithm for memory, to free when removing an instance.

A sample implementation of the delete function that calls `algNumAlloc()` and `algFree()` in sequence is provided in the `ALG_delete()` function implemented in the alg_create.c file.

## 3.2 Frame Buffer Management by Application

### 3.2.1 Frame Buffer Input and Output

With the new XDM, decoder does not ask for frame buffer at the time of `alg_create()`. It uses buffer from `XDM2_BufDesc *outBufs`, which it reads during each decode process call. Hence, there is no distinction between DPB and display buffers. The framework needs to ensure that it does not overwrite the buffers that are locked by the codec.

```
mp2VDEC_create();
mp2VDEC_control(XDM_GETBUFINFO); /* Returns default 1080p
HD size */
do{
mp2VDEC_decode(); //call the decode API
mp2VDEC_control(XDM_GETBUFINFO); /* updates the memory
required as per the size parsed in stream header */
}
while(all frames)
```

- Note:
- Application can take the information retured by the control function with the `XDM_GETBUFINFO` command and change the size of the buffer passed in the next process call.
- The output luma buffer size required is : `((width aligned to multiple of 16 +2)*(height aligned to multiple of 16 +2)`.
  For chroma buffer, the height need to be halved. This assumes worst case padding requirement, That is, for interlaced coding. For the first `GETBUFINFO` call, `maxheigh`t and `maxwidth` are used for calculating buffer size. For subsequent `GETBUFINFO` calls (That is, after the first process call, when the decoder gets to know the

actual height and width from the headers) the actual height and with are used. Hence, this can be optionally used by the application to re allocate the buffer sizes, if required.

- Application can re-use the extra buffer space of the 1st frame, if the above control call returns a small size than that was provided.

The frame pointer given by the application and that returned by the algorithm may be different. `BufferID` (`InputID/outputID`) provides the unique ID to keep a record of the buffer given to the algorithm and released by the algorithm.

As explained above, buffer pointer cannot be used as a unique identifier to keep a record of frame buffers. Any buffer given to algorithm should be considered locked by algorithm, unless the buffer is returned to the application through `IVIDDEC3_OutArgs->freeBufID[]`.

- Note:
- `BufferID` returned in `IVIDDEC3_OutArgs ->outputID[]` is only for display purpose. Application should not consider it free unless it is a part of `IVIDDEC3_OutArgs->freeBufID[]`.

### 3.2.2 Frame Buffer Format

The frame buffer format to be used for both progressive and interlaced pictures is as explained in Chapter 6 of this document.

### 3.2.3 Frame Buffer Management by Application

The application framework can efficiently manage frame buffers by keeping a pool of free frames from which it gives the decoder empty frames on request.

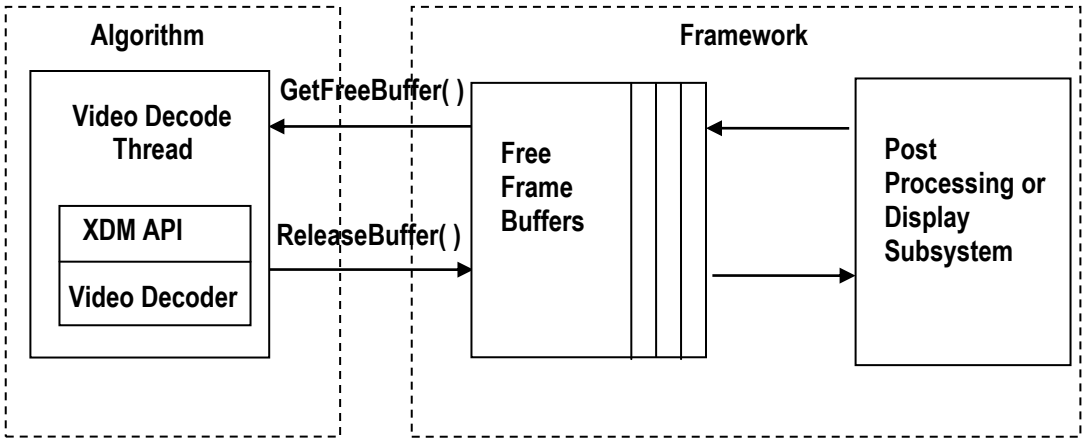



*Figure 3-3. Interaction of Frame Buffers Between Application and Framework*

The sample application also provides a prototype for managing frame buffers. It implements the following functions, which are defined in file buffermanager.c provided along with test application.

- `BUFFMGR_Init()` – `BUFFMGR_Init` function is called by the test application to initialize the global buffer element array to default and to allocate the required number of memory data for reference and output buffers. The maximum required DPB size is defined by the supported profile and level.
- `BUFFMGR_ReInit()` - `BUFFMGR_ReInit` function allocates global luma and chroma buffers and allocates entire space to the first element. This element will be used in the first frame decode. After the picture height and width and its luma and chroma buffer requirements are obtained, the global luma and chroma buffers are re-initialized to other elements in the buffer array.
- `BUFFMGR_GetFreeBuffer()` - `BUFFMGR_GetFreeBuffer` function searches for a free buffer in the global buffer array and returns the address of that element. Incase none of the elements are free, then it returns `NULL`.
- `BUFFMGR_ReleaseBuffer()` - `BUFFMGR_ReleaseBuffer` function takes an array of buffer-IDs which are released by the test application. 0 is not a valid buffer ID, hence this function moves until it encounters a buffer ID as zero or it hits the `MAX_BUFF_ELEMENTS`.
- `BUFFMGR_DeInit()`- `BUFFMGR_DeInit` function releases all memory allocated by buffer manager.

## 3.3 Handshaking Between Application and Algorithm

Application provides the algorithm with its implementation of functions for the video task to move to `SEM-pend` state, when the execution happens in the co-processor. The algorithm calls these application functions to move the video task to `SEM-pend` state.

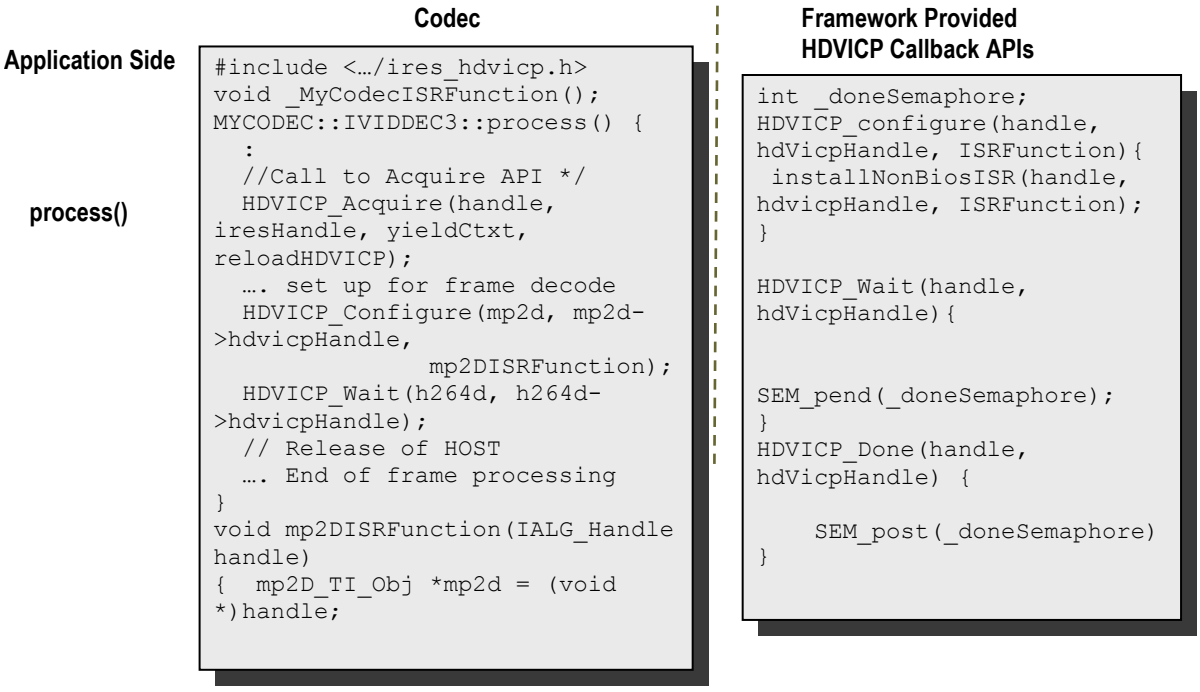



*Figure 3-4. Interaction Between Application and Codec*

- Note:
- Process call architecture to share Host resource among multiple threads.
- ISR ownership is with the Host layer resource manager – outside the codec.
- The actual codec routine to be executed during ISR is provided by the codec.
- OS/System related calls (`SEM_pend`, `SEM_post`) also outside the codec.
- Codec implementation is OS independent.

The functions to be implemented by the application are:

- `void HDVICP_Acquire(IALG_Handle handle, IRES_HDVICP2_Handle iresHandle, IRES_YieldContext * yieldCtxt, Bool *reloadHDVICP)`

  This function is called by the algorithm to acquire the HDVICP2 resource.

- `HDVICP_Configure(IALG_Handle handle, IRES_HDVICP2_Handle iresHandle, void(*IRES_HDVICP2_CallbackFxn)(IALG_Handle handle, void *cbArgs), void *cbArgs)`

This function is called by the algorithm to register its ISR function, which the application needs to call when it receives interrupts pertaining to the video task.

- `HDVICP_Wait (void *hdvicpHandle)`

  This function is called by the algorithm to move the video task to `SEM-pend` state.

- `HDVICP_Done (void *hdvicpHandle)`

  This function is called by the algorithm to release the video task from `SEM-pend` state. In the sample test application, these functions are implemented in hdvicp_framework.c file. The application can implement it in a way considering the underlying system.

## 3.4 Address Translations

The buffers addresses(DDR addresses) as seen by Media Controller and HDVICP2(VDMA) will be different. Hence, address translations are needed to convert from one address view to another. The application needs to implement a MEMUTILS function for this address translation (which will be later implemented by the framework components). An example of the address translation function is as shown. The codec will make a call to this function from the host (Media Controller) library. Therefore, the function name and arguments should follow the example provided below. For a given input address, this function returns the VDMA view of the buffer (that is, address as seen by HDVICP2).

```
void *MEMUTILS_getPhysicalAddr(Ptr Addr)
{
return ((void *)((unsigned int)Addr & VDMAVIEW_EXTMEM));
}
```

Sample settings for the macro `VDMAVIEW_EXTMEM` is as shown.

```
#if defined(HOSTARM968_FPGA)
     #define VDMAVIEW_EXTMEM (0x07FFFFFF)
#elif defined(HOSTCORTEXMEDIA CONTROLLER_OMAP4)
     #define VDMAVIEW_EXTMEM (0xFFFFFFFF)
#else
     #define VDMAVIEW_EXTMEM (0x07FFFFFF)
#endif
```

## 3.5 Sample Test Application

The test application exercises the `IVIDDEC3` base class of the MPEG-2 Decoder.

```
/*Main Function acting as a client for Video Decode Call*/

  BUFFMGR_Init();

  TestApp_SetInitParams(&params.viddecParams);

  /*---------------- Decoder creation -----------------*/
  handle = (IALG_Handle) mp2VDEC_create();
```

```
  /* Get Buffer information             */
  MPEG2VDEC_TI_control(handle, XDM_GETBUFINFO);

  /* Do-While Loop for Decode Call  for a given stream  */
    do
    {
/* Read the bitstream in the Application Input Buffer */
      validBytes = TestApp_ReadByteStream(inFile);

       /* Get free buffer from buffer pool */
        buffEle = BUFFMGR_GetFreeBuffer();
/* Optional: Set Run-time parameters in the Algorithm via
control() */

     MPEG2VDEC_TI_control(handle, XDM_SETPARAMS);

/*-----------------------------------------------------*/
/* Start the process : To start decoding a frame      */
/*-----------------------------------------------------*/
      retVal = MPEG2VDEC_TI_decode
            (
              handle,
              (XDM1_BufDesc *)&inputBufDesc,
              (XDM_BufDesc *)&outputBufDesc,
              (IVIDDEC3_InArgs *)&inArgs,
              (IVIDDEC3_OutArgs *)&outArgs
            );

     /* Get the statatus of the decoder using comtrol */
     MPEG2VDEC_TI_control(handle, IMPEG2VDEC_GETSTATUS);

     /* Get Buffer information             */
     MPEG2VDEC_TI_control(handle, XDM_GETBUFINFO);


      /* Optional: Reinit the buffer manager in case the
      /* frame size is different           */
     BUFFMGR_ReInit();

      /* Always release buffers - which are released from
      /* the algorithm side -back to the buffer manager
*/
      BUFFMGR_ReleaseBuffer((XDAS_UInt32
*)outArgs.freeBufID);


}  while(1);
/* end of Do-While loop - which decodes frames         */

ALG_delete (handle);

BUFFMGR_DeInit();
```

- Note:
- This sample test application does not depict the actual function parameter or control code. It shows the basic flow of the code.

Chapter 4

# API Reference

This chapter provides a detailed description of the data structures and interfaces functions used in the codec component.

**Topic** **Page**

## 4.1 Symbolic Constants and Enumerated Data Types

This section describes the XDM defined data structures that are common across codec classes. These XDM data structures can be extended to define any implementation specific parameters for a codec component.

*Table 4-1. List of Enumerated Data Types*

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| IVIDEO_FrameType | IVIDEO_NA_FRAME | Frame type not available |
| | IVIDEO_I_FRAME | Intra coded frame |
| | IVIDEO_P_FRAME | Forward inter coded frame |
| | IVIDEO_B_FRAME | Bi-directional inter coded frame |
| | IVIDEO_IDR_FRAME | Intra coded frame that can be used for refreshing video content. Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_II_FRAME | Interlaced Frame, both fields are I frames |
| | IVIDEO_IP_FRAME | Interlaced Frame, first field is an I frame, second field is a P frame |
| | IVIDEO_IB_FRAME | Interlaced Frame, first field is an I frame, second field is a B frame. Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_PI_FRAME | Interlaced Frame, first field is a P frame, second field is a I frame. Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_PP_FRAME | Interlaced Frame, both fields are P frames |
| | IVIDEO_PB_FRAME | Interlaced Frame, first field is a P frame, second field is a B frame. Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_BI_FRAME | Interlaced Frame, first field is a B frame, second field is an I frame. Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_BP_FRAME | Interlaced Frame, first field is a B frame, second field is a P frame. Not supported in this version of MPEG2 Decoder. |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | IVIDEO_BB_FRAME | Interlaced Frame, both fields are B frames. Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_MBAFF_I_FRAME | Intra coded MBAFF frame. Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_MBAFF_P_FRAME | Forward inter coded MBAFF frame. Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_MBAFF_B_FRAME | Bi-directional inter coded MBAFF frame. Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_MBAFF_IDR_FRAME | Intra coded MBAFF frame that can be used for refreshing video content. Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_FRAMETYPE_DEFAULT | Default set to IVIDEO_I_FRAME |
| IVIDEO_ContentType | IVIDEO_CONTENTTYPE_NA | Content type is not applicable |
| | IVIDEO_PROGRESSIVE IVIDEO_PROGRESSIVE_FRAME | Progressive video content |
| | IVIDEO_INTERLACED IVIDEO_INTERLACED_FRAME | Interlaced video content |
| | IVIDEO_INTERLACED_TOPFIELD | Interlaced video content, Top field |
| | IVIDEO_INTERLACED_BOTTOMFIELD | Interlaced video content, Bottom field |
| | IVIDEO_CONTENTTYPE_DEFAULT | Default set to IVIDEO_PROGRESSIVE |
| IVIDEO_FrameSkip | IVIDEO_NO_SKIP | Do not skip the current frame. Default Value. |
| | IVIDEO_SKIP_P | Skip forward inter coded frame. Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SKIP_B | Skip bi-directional inter coded frame. |
| | IVIDEO_SKIP_I | Skip intra coded frame. Not supported in this version of MPEG2 Decoder. |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | IVIDEO_SKIP_IP | Skip I and P frame/field(s). Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SKIP_IB | Skip I and B frame/field(s). Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SKIP_PB | Skip P and B frame/field(s). Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SKIP_IPB | Skip I/P/B/BI frames Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SKIP_IDR | Skip IDR Frame Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SKIP_NONREFERENCE | Skip non reference frame. |
| | IVIDEO_SKIP_DEFAULT | Default set to IVIDEO_NO_SKIP |
| IVIDEO_VideoLayout | IVIDEO_FIELD_INTERLEAVED | Buffer layout is interleaved. |
| | IVIDEO_FIELD_SEPARATED | Buffer layout is field separated. |
| | IVIDEO_TOP_ONLY | Buffer contains only top field. |
| | IVIDEO_BOTTOM_ONLY | Buffer contains only bottom field |
| IVIDEO_OperatingMode | IVIDEO_DECODE_ONLY | Decoding Mode |
| | IVIDEO_ENCODE_ONLY | Encoding Mode. Not applicable for decoder. |
| | IVIDEO_TRANSCODE_FRAMELEVEL | Transcode Mode of operation (encode/decode), which consumes /generates transcode information at the frame level. |
| | IVIDEO_TRANSCODE_MBLEVEL | Transcode Mode of operation (encode/decode), which consumes /generates transcode information at the MB level. Not supported in this version of decoder. |
| | IVIDEO_TRANSRATE_FRAMELEVEL | Transrate Mode of operation for encoder, which consumes transrate information at the frame level. Not supported in this version of decoder. |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | IVIDEO_TRANSRATE_MBLEVEL | Transrate Mode of operation for encoder, which consumes transrate information at the MB level. Not supported in this version of decoder. |
| IVIDEO_OutputFrameStatus | IVIDEO_FRAME_NOERROR | Output buffer is available. |
| | IVIDEO_FRAME_NOTAVAILABLE | Codec does not have any output buffers. |
| | IVIDEO_FRAME_ERROR | Output buffer is available and corrupted. |
| | IVIDEO_FRAME_OUTPUTSKIP | The video frame was skipped (that is not decoded) |
| | IVIDEO_OUTPUTFRAMESTATUS_DEFAULT | Default set to IVIDEO_FRAME_NOERROR |
| IVIDEO_PictureType | IVIDEO_NA_PICTURE | Frame type not available |
| | IVIDEO_I_PICTURE | Intra coded picture |
| | IVIDEO_P_PICTURE | Forward inter coded picture |
| | IVIDEO_B_PICTURE | Bi-directional inter coded picture |
| IVIDEO_DataMode | IVIDEO_FIXEDLENGTH | Input to the decoder is in multiples of a fixed length (example, 4K) (input side for decoder). Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SLICEMODE | Slice mode of operation (Input side for decoder).<br>Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_NUMROWS | Number of rows, each row is 16 lines of video (output side for decoder).<br>Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_ENTIREFRAME | Processing of entire frame data |
| IVIDDEC3_displayDelay | IVIDDEC3_DISPLAY_DELAY_AUTO | Decoder decides the display delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DECODE_ORDER | Display frames are in decoded order without delay |
| | IVIDDEC3_DISPLAY_DELAY_1 | Display the frames with 1 frame delay |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | IVIDDEC3_DISPLAY_DELAY_2 | Display the frames with 2 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_3 | Display the frames with 3 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_4 | Display the frames with 4 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_5 | Display the frames with 5 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_6 | Display the frames with 6 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_7 | Display the frames with 7 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_8 | Display the frames with 8 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_9 | Display the frames with 9 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_10 | Display the frames with 10 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_11 | Display the frames with 11 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_12 | Display the frames with 12 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_13 | Display the frames with 13 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_14 | Display the frames with 14 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAY_DELAY_15 | Display the frames with 15 frame delay. Not supported in this version of MPEG2 Decoder. |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | IVIDDEC3_DISPLAY_DELAY_16 | Display the frames with 16 frame delay. Not supported in this version of MPEG2 Decoder. |
| | IVIDDEC3_DISPLAYDELAY_DEFAULT | Same as IVIDDEC3_DISPLAY_DELAY_AUTO. Not supported in this version of MPEG2 Decoder. |
| XDM_DataFormat | XDM_BYTE | Big endian stream (default value) |
| | XDM_LE_16 | 16-bit little endian stream. Not supported in this version of MPEG2 Decoder. |
| | XDM_LE_32 | 32-bit little endian stream. Not supported in this version of MPEG2 Decoder. |
| | XDM_LE_64 | 64-bit little endian stream. Not supported in this version of MPEG2 Decoder. |
| | XDM_BE_16 | 16-bit big endian stream. Not supported in this version of MPEG2 Decoder. |
| | XDM_BE_32 | 32-bit big endian stream. Not supported in this version of MPEG2 Decoder. |
| | XDM_BE_64 | 64-bit big endian stream. Not supported in this version of MPEG2 Decoder. |
| XDM_ChromaFormat | XDM_YUV_420P | YUV 4:2:0 planar. Not supported in this version of MPEG2 Decoder. |
| | XDM_YUV_422P | YUV 4:2:2 planar. Not supported in this version of MPEG2 Decoder. |
| | XDM_YUV_422IBE | YUV 4:2:2 interleaved (big endian). Not supported in this version of MPEG2 Decoder. |
| | XDM_YUV_422ILE | YUV 4:2:2 interleaved (little endian) (default value). Not supported in this version of MPEG2 Decoder. |
| | XDM_YUV_444P | YUV 4:4:4 planar. Not supported in this version of MPEG2 Decoder. |
| | XDM_YUV_411P | YUV 4:1:1 planar. Not supported in this version of MPEG2 Decoder. |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | XDM_GRAY | Gray format.<br>Not supported in this version of MPEG2 Decoder. |
| | XDM_RGB | RGB color format.<br>Not supported in this version of MPEG2 Decoder. |
| | XDM_YUV_420SP | YUV 4:2:0 chroma semi-planar |
| | XDM_ARGB8888 | ARGB8888 color format.<br>Not supported in this version of MPEG2 Decoder. |
| | XDM_RGB555 | RGB555 color format.<br>Not supported in this version of MPEG2 Decoder. |
| | XDM_RGB565 | RGB565 color format.<br>Not supported in this version of MPEG2 Decoder. |
| | XDM_YUV_444ILE | YUV 4:4:4 interleaved (little endian) color format.<br>Not supported in this version of MPEG2 Decoder. |
| XDM_MemoryType | XDM_MEMTYPE_ROW | Raw Memory Type (deprecated) |
| | XDM_MEMTYPE_RAW | Raw Memory Type i.e., Linear (standard) memory. |
| | XDM_MEMTYPE_TILED8 | 2D memory in 8-bit container of tiled memory space. |
| | XDM_MEMTYPE_TILED16 | 2D memory in 16-bit container of tiled memory space. |
| | XDM_MEMTYPE_TILED32 | 2D memory in 32-bit container of tiled memory space. Not supported in this version of MPEG2 Decoder. |
| | XDM_MEMTYPE_TILEDPAGE | 2D memory in page container of tiled memory space. |
| XDM_CmdId | XDM_GETSTATUS | Query algorithm instance to fill `Status` structure |
| | XDM_SETPARAMS | Set run-time dynamic parameters via the `DynamicParams` structure |
| | XDM_RESET | Reset the algorithm. |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | XDM_SETDEFAULT | Initialize all fields in Params structure to default values specified in the library. |
| | XDM_FLUSH | Handle end of stream conditions. This command forces algorithm instance to output data without additional input. |
| | XDM_GETBUFINFO | Query algorithm instance regarding the properties of input and output buffers |
| | XDM_GETVERSION | Query the algorithm's version. The result will be returned in the data field of the Status structure. Application has to allocate memory for a buffer passed through data field. The minimum buffer size required is 96 bytes. |
| | XDM_GETCONTEXTINFO | Query a split codec part for its context needs. Not supported in this version of MPEG2 Decoder. |
| | XDM_GETDYNPARAMSDEFAULT | Query algorithm instance regarding the dynamic parameters default values. |
| | XDM_SETLATEACQUIREARG | Set an algorithm's 'late acquire' argument. |
| XDM_AccessMode | XDM_ACCESSMODE_READ | The algorithm read from the buffer using the CPU |
| | XDM_ACCESSMODE_WRITE | The algorithm wrote from the buffer using the CPU |
| XDM_ErrorBit | XDM_APPLIEDCONCEALMENT | Bit 9<br>1 - applied concealment<br>0 - Ignore |
| | XDM_INSUFFICIENTDATA | Bit 10<br>1 - Insufficient data<br>0 - Ignore |
| | XDM_CORRUPTEDDATA | Bit 11<br>1 - Data problem/corruption<br>0 - Ignore |
| | XDM_CORRUPTEDHEADER | Bit 12<br>1 - Header problem/corruption<br>0 - Ignore |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | XDM_UNSUPPORTEDINPUT | Bit 13<br>1 - Unsupported feature/parameter in input<br>0 - Ignore |
| | XDM_UNSUPPORTEDPARAM | Bit 14<br>1 - Unsupported input parameter or configuration<br>0 - Ignore |
| | XDM_FATALERROR | Bit 15<br>1 - Fatal error<br>0 - Recoverable error |
| IMPEG2VDEC_ExtendedErrorCodes | IMPEG2VDEC_ERR_UNSUPPORTED_VIDDEC3PARAMS | Bit 0<br><br>This error code has been deprecated. |
| | IMPEG2VDEC_ERR_UNSUPPORTED_VIDDEC3DYNAMICPARAMS | Bit 1<br>1 - Base class Dynamic params out of supported range<br>0 - Ignore |
| | IMPEG2VDEC_ERR_UNSUPPORTED_MPEG2DECDYNAMICPARAMS | Bit 2<br>1 - Extended class Dynamic params out of supported range<br>0 - Ignore |
| | IMPEG2VDEC_ERR_IMPROPER_DATASYNC_SETTING | Bit 3<br><br>This error code has been deprecated. |
| | IMPEG2VDEC_ERR_NOSLICE | Bit 4<br>1 - Illegal start code search error from ECD<br>0 - Ignore |
| | IMPEG2VDEC_ERR_SLICEHDR | Bit 5<br>1 - Illegal Slice start/End error from ECD<br>0 - Ignore |
| | IMPEG2VDEC_ERR_MBDATA | Bit 6<br>1 - Data error detected by ECD<br>0 - Recoverable error |
| | IMPEG2VDEC_ERR_UNSUPPFEATURE | Bit 7<br>1 - Unsupported header extensions and profile/levels<br>0 - Ignore |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | IMPEG2VDEC_ERR_STREAM_END | Bit 16<br>1 - End of stream Detected while parsing<br>0 - Ignore |
| | IMPEG2VDEC_ERR_UNSUPPRESOLUTION | Bit 17<br>1 - Stream resolution out of create time range<br>0 - Ignore |
| | IMPEG2VDEC_ERR_STANDBY | Bit 18<br>1 - HDVICP2 not in standby on acquire<br>0 - Ignore |
| | IMPEG2VDEC_ERR_INVALID_MBOX_MESSAGE | Bit 19<br>1 - Message from icont to host not defined<br>0 - Recoverable error |
| | IMPEG2VDEC_ERR_HDVICP_RESET | Bit 20<br>1 - HDVICP2 reset API from host to RMAN failed<br>0 - Ignore |
| | IMPEG2VDEC_ERR_HDVICP_WAIT_NOT_CLEAN_EXIT | Bit 21<br>1 - Erroneous exit from the wait API from host to RMAN<br>0 - Ignore |
| | IMPEG2VDEC_ERR_SEQHDR | Bit 22<br>1 - Invalid values of the parameters in sequence header<br>0 - Ignore |
| | IMPEG2VDEC_ERR_GOP_PICHDR | Bit 23<br>1 - Invalid values of the parameters in GOP and picture headers<br>0 - Ignore |
| | IMPEG2VDEC_ERR_SEQLVL_EXTN | Bit 24<br>1 - Invalid values of the parameters in sequence level extension headers<br>0 - Recoverable error |
| | IMPEG2VDEC_ERR_PICLVL_EXTN | Bit 25<br>1 - Invalid values of the parameters in picture level extension headers<br>0 - Ignore |
| | IMPEG2VDEC_ERR_TRICK_MODE | Bit 26<br>1 - Denotes frame skipped/ reference frame skipped in Trick mode<br>0 - Ignore |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | IMPEG2VDEC_ERR_PICSIZECHANGE | Bit 27<br>1 - Change in frame dimensions in repeat sequence header<br>0 - Ignore |
| | IMPEG2VDEC_ERR_SEMANTIC | Bit 28<br>1 - Semantic rules of a MPEG-2 stream not followed<br>0 - Ignore |
| | IMPEG2VDEC_ERR_DECODE_EXIT | Bit 29<br>1 - Decoder failed to consume all the input bytes / All Mbs not decoded. This has been added to detect the corruption of dimension parameters in header.<br>0 - Ignore |
| | IMPEG2VDEC_ERR_IRES_RESHANDLE | Bit 30<br>1 - Invalid resource handle in IRES interface<br>0 - Ignore |
| | IMPEG2VDEC_ERR_IRES_RESDESC | Bit 31<br><br>This error code has been deprecated. |
| Impeg2VDEC_ErrorStatus | MPEG2_ECD_ILLEGAL_EOM | Bit 0 of extendedErrorCode0<br><br>1 – ECD3 cannot find EOM, end of macroblock, at the end of macroblock when picture_type is D-picture<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_EOB | Bit 1 of extendedErrorCode0<br><br>1 - ECD3 cannot find EOB, end of block, in a 64 coefficient block<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_MP1_ESCAPE_LVL | Bit 2 of extendedErrorCode0<br><br>1 - Decoded level from MPEG-2 ESCAPE code is 0x000 or 0x800<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_MP2_ESCAPE_LVL | Bit 3 of extendedErrorCode0<br><br>1 - Decoded level from MPEG-1 ESCAPE code is 0x0000 or 0x8000<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_ECD_ILLEGAL_MARKER_CONCEAL | Bit 4 of extendedErrorCode0<br><br>1 - Invalid parameter value in data<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_MBTYPE_D_PIC | Bit 5 of extendedErrorCode0<br><br>1 - Invalid parameter value in data<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_DCT_COEFF | Bit 6 of extendedErrorCode0<br><br>1 - Invalid parameter value in data<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_CBP | Bit 7 of extendedErrorCode0<br><br>1 - Invalid parameter value in data<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_MOTION_CODE | Bit 8 of extendedErrorCode0<br><br>1 - Invalid parameter value in data<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_MB_TYPE | Bit 9 of extendedErrorCode0<br><br>1 – Invalid parameter value in data<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_MB_ADDR_INCR | Bit 10 of extendedErrorCode0<br><br>1 – Invalid parameter value in data<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_EOS | Bit 11 of extendedErrorCode0<br><br>1 – End of slice cannot be found at the end of picture<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_QUANT_SCALE_CODE | Bit 12 of extendedErrorCode0<br><br>1 – Decoded quantizer_scale_code is zero<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_SLICE_START_POS | Bit 13 of extendedErrorCode0<br><br>1 – Showing that two data is mismatched. ECD3 uses macroblock position in ECD3 ON registers and continue processing.<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_ECD_ILLEGAL_START_CODE_SEARCH | Bit 14 of extendedErrorCode0<br><br>1 – Showing next start code searching infinite error<br>0 - Error not found |
| | MPEG2_ECD_ILLEGAL_DC_COEFF_OVFL | Bit 15<br>1 – Result of DC prediction is overflow or underflow<br>0 - Error not found |
| | MPEG2_DYNAMIC_PARAMS_HANDLE_ERROR | Bit 16 of extendedErrorCode0<br><br>1 – Dynamic parameters handle NULL<br>0 - Error not found |
| | MPEG2_STATUS_HANDLE_ERROR | Bit 17 of extendedErrorCode0<br><br>This error code has been deprecated. |
| | MPEG2_DYNAMIC_PARAMS_SIZE_ERROR | Bit 18 of extendedErrorCode0<br><br>1 – Dynamic params size neither base class or extended class size<br>0 - Error not found |
| | MPEG2_STATUS_SIZE_ERROR | Bit 19 of extendedErrorCode0<br><br>This error code has been deprecated. |
| | MPEG2_DECODE_HEADER_ERROR | Bit 20 of extendedErrorCode0<br><br>1 – DynamicParams.decodeHeader value out of supported range<br>0 - Error not found |
| | MPEG2_DISPLAY_WIDTH_ERROR | Bit 21 of extendedErrorCode0<br><br>1 – DynamicParams.displayWidth value out of supported range<br>0 - Error not found |
| | MPEG2_FRAME_SKIP_MODE_ERROR | Bit 22 of extendedErrorCode0<br><br>1 – DynamicParams.frameSkipMode value out of supported range<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_NEW_FRAME_FLAG_ERROR | Bit 23 of extendedErrorCode0<br><br>1 – DynamicParams.newFrameFlag value out of supported range<br>0 - Error not found |
| | MPEG2_GOTO_IFRAME_ERROR | Bit 24 of extendedErrorCode0<br><br>1 – Extended Dynamic param gotoNextIFrame out of supported range<br>0 - Error not found |
| | MPEG2_SKIP_BFRAME_ERROR | Bit 25 of extendedErrorCode0<br><br>1 – Extended Dynamic param skipBFrame out of supported range<br>0 - Error not found |
| | MPEG2_SKIP_CURRENTFRAME_ERROR | Bit 26 of extendedErrorCode0<br><br>1 – Extended Dynamic param skipCurrFrame out of supported range<br>0 - Error not found |
| | MPEG2_SEEK_FRAMEEND_ERROR | Bit 27 of extendedErrorCode0<br><br>1 – Extended Dynamic param seekFrameEnd out of supported range<br>0 - Error not found |
| | MPEG2_NULL_STATUS_DATA_BUF | Bit 28 of extendedErrorCode0<br><br>1 – Data Buffer pointer in status structure NULL in GETVERSION control call<br>0 - Error not found |
| | MPEG2_INSUFFICIENT_STATUS_DATA_BUF | Bit 29 of extendedErrorCode0<br><br>1 – Data buffers within status structure less than 96 bytes during getversion control call<br>0 - Error not found |
| | MPEG2_NULL_INARGS_POINTER_ERROR | Bit 30 of extendedErrorCode0<br><br>1 – InArgs pointer in process call NULL<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_INARGS_SIZE_ERROR | Bit 31 of extendedErrorCode0<br><br>1 – InArgs size in process call neither base class size nor extended class size<br>0 - Error not found |
| | MPEG2_INVALID_INPUT_BYTES_ERROR | Bit 0 of extendedErrorCode1<br><br>1 – Non-positive value of the input bytes(pInArgs->numBytes) to process call<br>0 - Error not found |
| | MPEG2_INVALID_INPUT_ID_ERROR | Bit 1 of extendedErrorCode1<br><br>1 – The input ID of output buffer is 0<br>0 - Error not found |
| | MPEG2_DECODER_NOT_INITIALIZED_ERROR | Bit 2 of extendedErrorCode1<br><br>This error code has been deprecated. |
| | MPEG2_NULL_INPUT_BUF_DESC_ERROR | Bit 3 of extendedErrorCode1<br><br>1 – Pointer to the Input Buffer descriptor is NULL<br>0 - Error not found |
| | MPEG2_NULL_INPUT_BUFFER_POINTER_ERROR | Bit 4 of extendedErrorCode1<br><br>1 – Input buffer pointer is NULL<br>0 - Error not found |
| | MPEG2_INVALID_INPUT_BUFFER_SIZE_ERROR | Bit 5 of extendedErrorCode1<br><br>1 – Invalid Input buffer size: when non-positive values for bytes in raw and dimensions in tiler<br>0 - Error not found |
| | MPEG2_INVALID_NUM_OF_INPUT_BUFFERS_ERROR | Bit 6 of extendedErrorCode1<br><br>1 – Insufficient buffers (when number of Input Buffers less than 1)<br>0 - Error not found |
| | MPEG2_EXCESS_NUM_OF_INPUT_BUFFERS_ERROR | Bit 7 of extendedErrorCode1<br><br>1 – Redundant input buffers (greater than 1)<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_INVALID_INPUT_BUFFER_MEMTYPE_ERROR | Bit 8 of extendedErrorCode1<br><br>1 – Input stream buffer memory type other than RAW/Page mode<br>0 - Error not found |
| | MPEG2_NULL_OUTARGS_POINTER_ERROR | Bit 9 of extendedErrorCode1<br><br>1 – OutArgs pointer is NULL<br>0 - Error not found |
| | MPEG2_INVALID_OUTARGS_SIZE | Bit 10 of extendedErrorCode1<br><br>1 – OutArgs size set to value other than base class or extended class size<br>0 - Error not found |
| | MPEG2_NULL_OUTPUT_BUF_DESC_POINTER_ERROR | Bit 11 of extendedErrorCode1<br><br>1 – The Display buffer descriptor pointer in OutArgs is NULL<br>0 - Error not found |
| | MPEG2_NULL_OUTPUT_BUF_DESC_ERROR | Bit 12 of extendedErrorCode1<br><br>1 – The Output buffer descriptor pointer provided in process call is NULL<br>0 - Error not found |
| | MPEG2_INVALID_OUTPUT_BUFFER0_POINTER_ERROR | Bit 13 of extendedErrorCode1<br><br>1 – Luma output buffer pointer is NULL<br>0 - Error not found |
| | MPEG2_INVALID_OUTPUT_BUFFER0_SIZE_ERROR | Bit 14 of extendedErrorCode1<br><br>1 – Luma output buffer size/dimensions being non-positive<br>0 - Error not found |
| | MPEG2_INVALID_NUM_OF_OUTPUT_BUFFERS_ERROR | Bit 15 of extendedErrorCode1<br><br>1 – Number of Output buffers other than 2 (3 in metadataType Mbinfo)<br>0 - Error not found |
| | MPEG2_INVALID_OUTPUT_BUFFER0_MEMTYPE_ERROR | Bit 16 of extendedErrorCode1<br><br>1 – Luma buffer memory other than Raw, tiled8, and tiledpage mode<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_INVALID_OUTPUT_BUFFER0_ALIGNMENT_ERROR | Bit 17 of extendedErrorCode1<br><br>1 – Luma buffer base address not 128-bit aligned<br>0 - Error not found |
| | MPEG2_INVALID_OUTPUT_BUFFER1_POINTER_ERROR | Bit 18 of extendedErrorCode1<br><br>1 – Chroma output buffer pointer is NULL<br>0 - Error not found |
| | MPEG2_INVALID_OUTPUT_BUFFER1_SIZE_ERROR | Bit 19 of extendedErrorCode1<br><br>1 –Chroma output buffer size/dimensions being non-positive<br>0 - Error not found |
| | MPEG2_INVALID_OUTPUT_BUFFER1_MEMTYPE_ERROR | Bit 20 of extendedErrorCode1<br><br>1 – Chroma buffer memory other than Raw, tiled 8, tiled16 and page mode<br>0 - Error not found |
| | MPEG2_INVALID_OUTPUT_BUFFER1_ALIGNMENT_ERROR | Bit 21 of extendedErrorCode1<br><br>1 – Chroma buffer base address not 128-bit aligned<br>0 - Error not found |
| | MPEG2_INVALID_OUTPUT_BUFFER2_POINTER_ERROR | Bit 22 of extendedErrorCode1<br><br>1 – MBinfo output buffer pointer is NULL in metadataType Mbinfo mode<br>0 - Error not found |
| | MPEG2_INVALID_OUTPUT_BUFFER2_SIZE_ERROR | Bit 23 of extendedErrorCode1<br><br>1 – MBinfo output buffer size is non-positive in metadataType Mbinfo mode<br>0 - Error not found |
| | MPEG2_INVALID_OUTPUT_BUFFER2_MEMTYPE_ERROR | Bit 24 of extendedErrorCode1<br><br>1 – MBinfo output buffer memory type other than RAW in metadataType Mbinfo mode<br>0 - Error not found |
| | MPEG2_INVALID_BUFFER_USAGE_MODE | Bit 25 of extendedErrorCode1<br><br>This error code has been deprecated. |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_SEQ_HDR_INVALID_FRAME_WIDTH | Bit 26 of extendedErrorCode1<br><br>1 – Frame width from header being 0 or greater than max width provided at create time<br>0 - Error not found |
| | MPEG2_SEQ_HDR_INVALID_FRAME_HEIGHT | Bit 27 of extendedErrorCode1<br><br>1 – Frame height from header being 0 or greater than max height provided at create time<br>0 - Error not found |
| | MPEG2_SEQ_HDR_INVALID_ASPECT_RATIO | Bit 28 of extendedErrorCode1<br><br>1 – Aspect ratio from sequence header is not standard compliant<br>0 - Error not found |
| | MPEG2_SEQ_HDR_INVALID_FRAME_RATE_CODE | Bit 29 of extendedErrorCode1<br><br>1 – Frame rate code from sequence header not standard compliant<br>0 - Error not found |
| | MPEG2_INVALID_INTRA_QUANT_MAT | Bit 30 of extendedErrorCode1<br><br>1 – Intra quantization matrix in sequence header or quantization matrix extension is not standard compliant<br>0 - Error not found |
| | MPEG2_INVALID_NON_INTRA_QUANT_MAT | Bit 31 of extendedErrorCode1<br><br>1 – Non-Intra quantization matrix in sequence header or quantization matrix extension is not standard compliant<br>0 - Error not found |
| | MPEG2_SEQ_HDR_INVALID_INTRA_ESCAPE_BIT | Bit 0 of extendedErrorCode2<br><br>1 – Escape bit in ProfileandLevel indication of sequence extension is not standard compliant<br>0 - Error not found |
| | MPEG2_SEQ_HDR_INVALID_PROFILE | Bit 1 of extendedErrorCode2<br><br>1 – Unsupported Profile, other than simple and main<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_SEQ_HDR_INVALID_LEVEL | Bit 2 of extendedErrorCode2<br><br>1 – Invalid level indication in sequence extension<br>0 - Error not found |
| | MPEG2_SEQ_HDR_INVALID_RESOLUTION_FORLVL | Bit 3 of extendedErrorCode2<br><br>1 – Level based resolution constraints violated<br>0 - Error not found |
| | MPEG2_SEQ_HDR_INVALID_CHROMA_FORMAT | Bit 4 of extendedErrorCode2<br><br>1 – Chroma format in sequence extension other than 4:2:0<br>0 - Error not found |
| | MPEG2_SEQ_HDR_INVALID_LOW_DELAY | Bit 5 of extendedErrorCode2<br><br>1 – Low-delay flag set to '1' in sequence extension not supported<br>0 - Error not found |
| | MPEG2_SEQ_DSP_INVALID_VIDEO_FORMAT | Bit 6 of extendedErrorCode2<br><br>1 – Video format parameter in sequence display extension not standard compliant<br>0 - Error not found |
| | MPEG2_SEQ_DSP_INVALID_COLOUR_PRIM | Bit 7 of extendedErrorCode2<br><br>1 – Colour primaries parameter in sequence display extension not standard compliant<br>0 - Error not found |
| | MPEG2_SEQ_DSP_INVALID_TRF_CHARS | Bit 8 of extendedErrorCode2<br><br>1 – Transfer characteristics parameter in sequence display extension not standard compliant<br>0 - Error not found |
| | MPEG2_SEQ_DSP_INVALID_MAT_COEFFS | Bit 9 of extendedErrorCode2<br><br>1 – Matrix coefficients for colour conversion in sequence display extension not standard compliant<br>0 - Error not found |
| | MPEG2_GOP_HDR_INVALID_DROP_FLAG | Bit 10 of extendedErrorCode2<br><br>1 – The drop flag in GOP header set when frame rate is not 29.97 Hz<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_GOP_HDR_INVALID_HOUR | Bit 11 of extendedErrorCode2<br><br>1 – Invalid 'hour' parameter in GOP header<br>0 - Error not found |
| | MPEG2_GOP_HDR_INVALID_MIN | Bit 12 of extendedErrorCode2<br><br>1 – Invalid 'minute' parameter in GOP header<br>0 - Error not found |
| | MPEG2_GOP_HDR_INVALID_SEC | Bit 13 of extendedErrorCode2<br><br>1 – Invalid 'second' parameter in GOP header<br>0 - Error not found |
| | MPEG2_GOP_HDR_INVALID_TIME_CODE_PICTURES | Bit 14 of extendedErrorCode2<br><br>1 – Invalid 'frame' parameter in GOP header<br>0 - Error not found |
| | MPEG2_GOP_HDR_INVALID_BROKEN_LINK | Bit 15 of extendedErrorCode2<br><br>1 – broken link in GOP header for a closed GOP<br>0 - Error not found |
| | MPEG2_PIC_HDR_INVALID_TEMP_REF | Bit 16 of extendedErrorCode2<br><br>This error code has been deprecated. |
| | MPEG2_PIC_HDR_INVALID_PIC_TYPE | Bit 17 of extendedErrorCode2<br><br>1 – Invalid picture type code: D-picture in MPEG2 and B-picture in MPEG1 simple profile.<br>0 - Error not found |
| | MPEG2_PIC_HDR_INVALID_VBV_DELAY | Bit 18 of extendedErrorCode2<br><br>This error code has been deprecated. |
| | MPEG1_PIC_HDR_INVALID_FWD_FCODE | Bit 19 of extendedErrorCode2<br><br>1 – Forward f-code in picture header for B/P pictures is '0'<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG1_PIC_HDR_INVALID_BWD_FCODE | Bit 20 of extendedErrorCode2<br><br>1 – Backward f-code in picture header for B pictures is '0'<br>0 - Error not found |
| | MPEG2_PIC_HDR_INVALID_FCODE | Bit 21 of extendedErrorCode2<br><br>1 – Invalid fcode values in picture extension<br>0 - Error not found |
| | MPEG2_PIC_HDR_INVALID_PIC_STRUCTURE | Bit 22 of extendedErrorCode2<br><br>1 – Invalid picture_structure value in picture extension. Also, progressive sequence has to contain 'frame' pictures.<br>0 - Error not found |
| | MPEG2_PIC_HDR_INVALID_FIELD_COMB | Bit 23 of extendedErrorCode2<br><br>1 – Picture type combinations for filed pictures not standard compliant. Valid types: II, IP, PP, BB<br>0 - Error not found |
| | MPEG2_PIC_HDR_INVALID_TFF | Bit 24 of extendedErrorCode2<br><br>1 – Top_field_first flag in picture coding extension not standard compliant<br>0 - Error not found |
| | MPEG2_PIC_HDR_INVALID_FPFD | Bit 25 of extendedErrorCode2<br><br>1 – framePredFrameDct flag in picture coding extension not standard compliant: has to be '1' for progressive sequence; has to be '0' for field pictures<br>0 - Error not found |
| | MPEG2_PIC_HDR_INVALID_RFF | Bit 26 of extendedErrorCode2<br><br>1 – RepeatFirstField flag in picture coding extension had to be '0' for a progressive sequence<br>0 - Error not found |
| | MPEG2_PIC_HDR_INVALID_PROG_FLAG | Bit 27 of extendedErrorCode2<br><br>1 – Picture structure has to be 'frame' in a progressive frame<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_QUANT_EXT_INVALID_LOAD_CHROMA_INTRA_FLAG | Bit 28 of extendedErrorCode2<br><br>1 – load_chroma_intra_quantizer_matrix shall be '0' for 4:2:0 format<br>0 - Error not found |
| | MPEG2_QUANT_EXT_INVALID_LOAD_CHROMA_NON_INTRA_FLAG | Bit 29 of extendedErrorCode2<br><br>1 – load_chroma_non_intra_quantizer_matrix shall be '0' for 4:2:0 format<br>0 - Error not found |
| | MPEG2_INVALID_EXTN_CODE | Bit 30 of extendedErrorCode2<br><br>1 – Undefined header extension code<br>0 - Error not found |
| | MPEG2_SEQ_HDR_MISSING | Bit 31 of extendedErrorCode2<br><br>1 – Picture header is encountered before sequence header<br>0 - Error not found |
| | MPEG2_NO_PICTURE_ENCODED_ERROR | Bit 0 of extendedErrorCode3<br><br>This error code has been deprecated. |
| | MPEG2_SEQ_EXT_MISSING | Bit 1 of extendedErrorCode3<br><br>1 – Occurrence of picture coding extension without sequence extension<br>0 - Error not found |
| | MPEG2_PIC_CODING_EXT_MISSING | Bit 2 of extendedErrorCode3<br><br>This error code has been deprecated. |
| | MPEG2_SEQ_DISP_EXT_MISSING | Bit 3 of extendedErrorCode3<br><br>This error code has been deprecated. |
| | MPEG2_GOP_FIRST_FRAME_NOT_I | Bit 4 of extendedErrorCode3<br><br>1 – First picture of a GOP not an I picture<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_SCALABILITY_NOT_SUPPORTED | Bit 5 of extendedErrorCode3<br><br>1 – Scalable extensions not supported, hence just parsed<br>0 - Error not found |
| | MPEG2_END_OF_SEQ_DETECTED | Bit 6 of extendedErrorCode3<br><br>1 – Sequence end code detected on parsing<br>0 - Error not found |
| | MPEG2_PIC_HDR_RFF_FRAME_RATE_MISMATCH | Bit 7 of extendedErrorCode3<br><br>1 – Standard defined dependency of repeatFirstField flag on frame rate violated<br>0 - Error not found |
| | MPEG2_PIC_HDR_INVALID_DC_PRECISION | Bit 8 of extendedErrorCode3<br><br>1 – intraDcPrecision decides the scaling factor in the inverse quantization process for the DC coefficient of an intra block. This has an invalid value.<br>0 - Error not found |
| | MPEG2_INVALID_FRAME_RATE | Bit 9 of extendedErrorCode3<br><br>1 – Frame rate from header is '0' or greater than the max value set during create time<br>0 - Error not found |
| | MPEG2_INVALID_BIT_RATE | Bit 10 of extendedErrorCode3<br><br>1 – Bit-rate from header is '0' or greater than the max value set during create time<br>0 - Error not found |
| | MPEG2_FRAME_SKIPPED | Bit 11 of extendedErrorCode3<br><br>1 – Indicates that the current picture has been skipped from decoding in trick mode<br>0 - Error not found |
| | MPEG2_REF_FRAME_SKIPPED | Bit 12 of extendedErrorCode3<br><br>1 – Indicates that the reference picture for P/B piture has been skipped under trick mode<br>0 - Error not found |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|---|---|---|
| | MPEG2_NO_REF_TO_FLUSH | Bit 13 of extendedErrorCode3<br><br>1 – No more reference pictures held by the codec to provide in the flush call<br>0 - Error not found |
| | MPEG2_EXCESS_INPUT_BYTES | Bit 14 of extendedErrorCode3<br><br>1 – Entire input bytes have not been consumed by the codec<br>0 - Error not found |
| | MPEG2_ALL_MBS_NOT_DECODED | Bit 15 of extendedErrorCode3<br><br>1 – Number of decoded MBs is lesser than the expected number of MBs in the picture<br>0 - Error not found |
| XDM_MemoryUsageMode | XDM_MEMUSAGE_DATASYNC | Bit 0 - Data Sync mode. If this bit is set, the memory will be used in data sync mode.<br>Data sync feature is not supported in this version of MPEG-2 decoder. |

## 4.2 Data Structures

This section describes the XDM defined data structures, which are common across codec classes. These XDM data structures can be extended to define any implementation specific parameters for a codec component.

### *4.2.1 Common XDM Data Structures*

This section includes the following common XDM data structures:

- XDM2_SingleBufDesc
- XDM2_BufDesc
- XDM1_AlgBufInfo
- IVIDEO2_BufDesc
- IVIDDEC3_Fxns
- IVIDDEC3_Params
- IVIDDEC3_DynamicParams
- IVIDDEC3_InArgs
- IVIDDEC3_Status
- IVIDDEC3_OutArgs

#### 4.2.1.1 XDM2_SingleBufDesc

**Description**

This structure defines the buffer descriptor for single input and output buffers.

**Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| *buf | XDAS_Int8 | Input | Pointer to the buffer |
| memType | XDAS_Int16 | Input | Type of memory. See XDM_MemoryType enumeration for more details. |
| usageMode | XDAS_Int16 | Input | Memory usage descriptor. |
| bufSize | XDM2_BufSize | Input | Size of the buffer(for tile memory/row memory) |
| accessMask | XDAS_Int32 | Output | If the buffer was not accessed by the algorithm processor (for example, it was filled by DMA or other hardware accelerator that does not write through the algorithm CPU), then bits in this mask should not be set. |

#### 4.2.1.2 XDM2_BufSize

**Description**

This defines the union describing a buffer size.

**Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| width | XDAS_Int32 | Input | Width of buffer in 8-bit bytes.<br>Required only for tiled memory. |
| height | XDAS_Int32 | Input | Height of buffer in 8-bit bytes.<br>Required only for tiled memory. |
| bytes | XDM2_BufSize | Input | Size of the buffer in bytes |

#### 4.2.1.3 XDM2_BufDesc

**Description**

This structure defines the buffer descriptor for output buffers.

**Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| numBufs | XDAS_Int32 | Input | Number of buffers |
| descs[XDM_MAX_IO_BUFFERS] | XDM2_SingleBufDesc | Input | Array of buffer descriptors |

#### *4.2.1.4 XDM1_AlgBufInfo*

**‖ Description**

This structure defines the buffer information descriptor for input and output buffers. This structure is filled when you invoke the `control()` function with the `XDM_GETBUFINFO` command.

**‖ Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| minNumInBufs | XDAS_Int32 | Output | Number of input buffers |
| minNumOutBufs | XDAS_Int32 | Output | Number of output buffers |
| minInBufSize[XDM_MAX_IO_BUFFERS] | XDM2_BufSize | Output | Size required for each input buffer |
| minOutBufSize[XDM_MAX_IO_BUFFERS] | XDM2_BufSize | Output | Size required for each output buffer |
| inBufMemoryType[XDM_MAX_IO_BUFFERS] | XDAS_Int32 | Output | Memory type for each input buffer |
| outBufMemoryType[XDM_MAX_IO_BUFFERS] | XDAS_Int32 | Output | Memory type for each output buffer |
| minNumBufSets | XDAS_Int32 | Output | Minimum number of buffer sets for buffer management |

**Note:**

- For MPEG2 Main Profile Decoder, the buffer details are:
- Number of input buffer required is 1.
- Number of output buffers required is 2 if no metadata is requested by the application (one for Y plane and 1 for Cb/Cr plane).
- For frame mode of operation, there is no restriction on input buffer size except that it should contain atleast one frame of encoded data.
- The output buffer sizes (in bytes) for a resolution (X,Y) are
  - Y buffer = ((ROUND_TO_NEXT_MULT16(X)+2) * (ROUND_TO_NEXT_MULT16(Y)+2))

- UV buffer = ((ROUND_TO_NEXT_MULT16(X)+2) * (ROUND_TO_NEXT_MULT16(Y)/2 +2))
- Example: Output buffer sizes required for 1080p are
- Y buffer = (1920 + 2) * (1088 + 2)
  UV buffer = (1920 + 2) * (544 + 2)

- These are the maximum buffer sizes but they can be reconfigured depending on the format of the bit-stream.
- The memory types supported for input buffers are `XDM_MEMTYPE_RAW` and `XDM_MEMTYPE_TILEDPAGE`.
- The memory types supported for luma output buffers are `XDM_MEMTYPE_TILED8`, `XDM_MEMTYPE_TILEDPAGE` and `XDM_MEMTYPE_RAW`.
- The memory types supported for chroma output buffers are `XDM_MEMTYPE_TILED8`, `XDM_MEMTYPE_TILED16`, `XDM_MEMTYPE_TILEDPAGE` and `XDM_MEMTYPE_RAW`.

#### *4.2.1.5 IVIDEO2_BufDesc*

**‖ Description**

This structure defines the buffer descriptor for input and output buffers.

**‖ Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| `numPlanes` | `XDAS_Int32` | Input/Output | Number of buffers for video planes |
| `numMetaPlanes` | `XDAS_Int32` | Input/Output | Number of buffers for Metadata |
| `dataLayout` | `XDAS_Int32` | Input/ Output | Video buffer layout. See `IVIDEO_VideoLayout` enumeration for more details |
| `planeDesc [IVIDEO_MAX_NUM_PLANES]` | `XDM1_SingleBufDesc` | Input/ Output | Description for video planes |
| `metadataPlaneDesc [IVIDEO_MAX_NUM_METADATA_PLANES]` | `XDM1_SingleBufDesc` | Input/ Output | Description for metadata planes |
| `secondFieldOffsetWidth[IVIDEO_MAX_NUM_PLANES]` | `XDAS_Int32` | Input/ Output | Offset value for second field in `planeDesc` buffer (width in pixels) |
| `secondFieldOffsetHeight[IVIDEO_MAX_NUM_PLANES]` | `XDAS_Int32` | Input/ Output | Offset value for second field in `planeDesc` buffer (height in lines) |
| `imagePitch` | `XDAS_Int32 []` | Input/ Output | Image pitch for each plane |
| `imageRegion` | `XDM_Rect` | Input/ | Decoded image region including |

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| | | Output | padding /encoder input image |
| activeFrameRegion | XDM_Rect | Input/ Output | Actual display region/capture region |
| extendedError | XDAS_Int32 | Input/ Output | Provision for informing the error type if any |
| frameType | XDAS_Int32 | Input/ Output | Video frame types. See enumeration IVIDEO_FrameType. Not applicable for encoders |
| topFieldFirstFlag | XDAS_Int32 | Input/ Output | Indicates when the application (should display)/(had captured) the top field first. Not applicable for progressive content. |
| repeatFirstFieldFlag | XDAS_Int32 | Input/ Output | Indicates when the first field should be repeated. |
| frameStatus | XDAS_Int32 | Input/ Output | Video in/out buffer status. |
| repeatFrame | XDAS_Int32 | Input/ Output | Number of times to repeat the displayed frame. |
| contentType | XDAS_Int32 | Input/ Output | Video content type. See IVIDEO_ContentType. |
| chromaFormat | XDAS_Int32 | Input/ Output | Chroma format for encoder input data/decoded output buffer. See XDM_ChromaFormat enumeration for details. |
| scalingWidth | XDAS_Int32 | Input/ Output | Scaled image width for post processing for decoder. |
| scalingHeight | XDAS_Int32 | Input/ Output | Scaled image height for post processing for decoder. |
| rangeMappingLuma | XDAS_Int32 | Input/ Output | Applicable for VC1, set to -1 as default for other codecs |
| rangeMappingChroma | XDAS_Int32 | Input/ Output | Applicable for VC1, set to -1 as default for other codecs |
| enableRangeReductionFlag | XDAS_Int32 | Input/ Output | ON/OFF, default is OFF. Applicable only for VC1. |

**Note:**

- IVIDEO_MAX_NUM_PLANES: Max YUV buffers - one each for Y, U, and V.
- The following parameters are not supported/updated in this version of the decoder
  - scalingWidth
  - scalingHeight
  - rangeMappingLuma
  - rangeMappingChroma
  - enableRangeReductionFlag

#### 4.2.1.6 IVIDDEC3_Fxns

**Description**

This structure contains pointers to all the XDAIS and XDM interface functions.

**Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| ialg | IALG_Fxns | Input | Structure containing pointers to all the XDAIS interface functions.<br><br>For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360). |
| *process | XDAS_Int32 | Input | Pointer to the process() function |
| *control | XDAS_Int32 | Input | Pointer to the control() function |

#### 4.2.1.7 IVIDDEC3_Params

**Description**

This structure defines the creation parameters for an algorithm instance object. Set this data structure to NULL, if you are not sure of the values to be specified for these parameters.

**Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| size | XDAS_Int32 | Input | Size of the basic or extended (if being used) data structure in bytes. |

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| maxHeight | XDAS_Int32 | Input | Maximum video height to be supported in pixels. The supported range is [64, 2048]. Default is 1088. |
| maxWidth | XDAS_Int32 | Input | Maximum video width to be supported in pixels. The supported range is [64, 2048]. Default is 1920. |
| maxFrameRate | XDAS_Int32 | Input | Maximum frame rate in fps * 1000 to be supported.<br>Default is 30000. |
| maxBitRate | XDAS_Int32 | Input | Maximum bit-rate to be supported in bits per second. For example, if bit-rate is 10 Mbps, set this field to 10485760.<br>Default is 10000000. |
| dataEndianness | XDAS_Int32 | Input | Should be set to XDM_BYTE. |
| forceChromaFormat | XDAS_Int32 | Input | Sets the output to the specified format. Only 420 semi-planar format supported.<br>Default is XDM_YUV_420SP.<br><br>See XDM_ChromaFormat and eChromaFormat_t enumerations for details. |
| operatingMode | XDAS_Int32 | Input | Video coding mode of operation (encode/decode/transcode/transrate).<br>Only decode and transcode modes are supported in this version. Default is IVIDEO_DECODE_ONLY. |
| displayDelay | XDAS_Int32 | Input | Display delay to start display. Supported values are IVIDDEC3_DISPLAY_DELAY_AUTO, IVIDDEC3_DECODE_ORDER and IVIDDEC3_DISPLAY_DELAY_1.<br>Default value is IVIDDEC3_DISPLAY_DELAY_1. |
| inputDataMode | XDAS_Int32 | Input | Input mode of operation.<br>For decoder, it is fixed length/slice mode/entire frame.<br>This version of the decoder supports only the entire frame mode. |
| outputDataMode | XDAS_Int32 | Input | Output mode of operation.<br>For decoder, it is row mode/entire frame.<br>This version of the decoder supports only the entire frame mode. |
| numInputDataUnits | XDAS_Int32 | Input | Number of input slices/rows.<br>Not applicable for this MPEG-2 Decoder as it does not support sub-frame data synchronization. |

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| numOutputDataUnits | XDAS_Int32 | Input | Number of output slices/rows. Not applicable for this MPEG-2 Decoder as it does not support sub-frame data synchronization. |
| errorInfoMode | XDAS_Int32 | Input | Enable/disable packet error information for input/output. Only IVIDEO_ERRORINFO_OFF is supported. |
| displayBufsMode | XDAS_Int32 | Input | Indicates the displayBufs mode. This field can be set either as IVIDDEC3_DISPLAYBUFS_EMBEDDED or IVIDDEC3_DISPLAYBUFS_PTRS. Default is IVIDDEC3_DISPLAYBUFS_PTRS. |
| metadataType | XDAS_Int32[] | Input | Type of each metadata plane. This field can be set to either IVIDEO_METADATAPLANE_MBINFO or IVIDEO_METADATAPLANE_NONE. Default is IVIDEO_METADATAPLANE_NONE. |

**Note:**

- MPEG2 Decoder does not use the maxFrameRate and maxBitRate fields for creating the algorithm instance. In the current implementation, maxFrameRate is set to 1000 * 30, and maxBitRate is set to 10000000.
- Maximum video height and width supported are 2048 pixels and 2048 pixels respectively.
- The minimum height and width supported is 64 pixels.
- dataEndianness field should be set to XDM_BYTE.
- The default value of displayDelay is 1.
- InputDataMode can be set to IVIDEO_ENTIREFRAME only.

#### *4.2.1.8 IVIDDEC3_DynamicParams*

**Description**

This structure defines the run-time parameters for an algorithm instance object. Set this data structure to NULL, if you are not sure of the values to be specified for these parameters.

**Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| size | XDAS_Int32 | Input | Size of the basic or extended (if being used) data structure in bytes. |
| decodeHeader | XDAS_Int32 | Input | Number of access units to decode:<br>0 (XDM_DECODE_AU) - Decode entire frame including all the headers<br>1 (XDM_PARSE_HEADER) - Decode only one NAL unit or Parse the header only<br>Default is XDM_DECODE_AU. |
| displayWidth | XDAS_Int32 | Input | If the field is set to:<br>0 - Uses decoded image width as pitch<br>If any other value greater than the decoded image width is given, then this value in pixels is used as pitch.<br><br>When the output buffer is in non-TILED region, display width should be multiple of 128. Default value is 0. |
| frameSkipMode | XDAS_Int32 | Input | Frame skip mode. See IVIDEO_FrameSkip enumeration for details. |
| newFrameFlag | XDAS_Int32 | Input | Flag to indicate that the algorithm should start a new frame.<br>Valid values are XDAS_TRUE and XDAS_FALSE.<br>This is useful for error recovery, for example, when the end of frame cannot be detected by the codec but is known to the application. Not supported in this decoder. |
| *putDataFxn | XDM_DataSyncPutFxn | Input | DataSync call back function pointer for putData. Not supported in this decoder. |
| putDataHandle | XDM_DataSyncHandle | Input | DataSync handle for putData. Not supported in this decoder. |
| *getDataFxn | XDM_DataSyncGetFxn | Input | DataSync call back function pointer for getData. Not supported in this decoder. |
| getDataHandle | XDM_DataSyncHandle | Input | DataSync handle for getData. Not supported in this decoder. |
| putBufferFxn | XDM_DataSyncPutBufferFxn | Input | Not supported in this decoder. |
| putBufferHandle | XDM_DataSyncHandle | Input | Not supported in this decoder. |

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| lateAcquireArg | XDAS_Int32 | Input | Argument used during late acquire. Default value is IRES_HDVICP2_UNKNOWNLATEACQUIREARG. |

**Note:**

- The decoder supports displayWidth >=0. A value of 0 indicated the decoder will choose the displaywidth (displaywidth = imagewidth). The displayWidth should be >= image width. The default value of displayWidth is 0.
- MPEG2 Decoder does not support newFrameFlag. Their values are set as zero.
- In case of XDM_PARSE_HEADER, the application should provide a minimum of Sequence header including sequence extensions (if any) of the stream as input to the decoder.

#### *4.2.1.9 IVIDDEC3_InArgs*

**Description**

This structure defines the run-time input arguments for an algorithm instance object.

**Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| size | XDAS_Int32 | Input | Size of the basic or extended (if being used) data structure in bytes. |
| numBytes | XDAS_Int32 | Input | Size of input data (in bytes) provided to the algorithm for decoding |
| inputID | XDAS_Int32 | Input | Application passes this ID to algorithm and decoder will attach this ID to the corresponding output frames. This is useful in case of re-ordering (for example, B frames). If there is no re-ordering, outputID field in the IVIDDEC3_OutArgs data structure will be same as inputID field. |

**Note:**

MPEG2 Decoder copies the inputID value to the outputID value of IVIDDEC3_OutArgs structure after factoring in the display delay.

#### 4.2.1.10 IVIDDEC3_Status

**‖ Description**

This structure defines parameters that describe the status of an algorithm instance object.

**‖ Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| Size | XDAS_Int32 | Input | Size of the basic or extended (if being used) data structure in bytes. |
| extendedError | XDAS_Int32 | Output | Extended error code. See XDM_ErrorBit enumeration for details. |
| data | XDM1_SingleBufDesc | Output | Buffer information structure for information passing buffer. |
| maxNumDisplayBufs | XDAS_Int32 | Output | Maximum number of buffers required by the codec. |
| maxOutArgsDisplayBufs | XDAS_Int32 | Output | The maximum number of display buffers that can be returned through IVIDDEC3_OutArgs.displayBufs. |
| outputHeight | XDAS_Int32 | Output | Output height in pixels |
| outputWidth | XDAS_Int32 | Output | Output width in pixels |
| frameRate | XDAS_Int32 | Output | This value will be derived from VUI parameters as, frameRate = time_scale / (2 * num_units_in_ticks). In case the VUI parameters are absent, the frameRate will be reported as 0, which should be inferred as 'not available'. |
| bitRate | XDAS_Int32 | Output | Average bit-rate in bits per second |
| contentType | XDAS_Int32 | Output | Video content. See IVIDEO_ContentType enumeration for details. |
| sampleAspectRatioHeight | XDAS_Int32 | Output | Sample aspect ratio for height |
| sampleAspectRatioWidth | XDAS_Int32 | Output | Sample aspect ratio for width |
| bitRange | XDAS_Int32 | Output | Bit range. It is set to IVIDEO_YUVRANGE_FULL. |

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| forceChromaFormat | XDAS_Int32 | Output | Output chroma format. See XDM_ChromaFormat and eChromaFormat_t enumeration for details. |
| operatingMode | XDAS_Int32 | Output | Mode of operation: Encoder/Decoder/Transcode/Transrate. This decoder supports IVIDEO_DECODE_ONLY and IVIDEO_TRANSCODE_FRAMELEVEL only. |
| frameOrder | XDAS_Int32 | Output | Indicates the output frame order. See IVIDDEC3_displayDelay enumeration for more details. |
| inputDataMode | XDAS_Int32 | Output | Input mode of operation. For decoder, it is fixed length/slice mode/entire frame. This version of the decoder supports only the entire frame mode. |
| outputDataMode | XDAS_Int32 | Output | Output mode of operation. For decoder, it is the row mode/entire frame. This version of the decoder supports only the entire frame mode. |
| bufInfo | XDM_AlgBufInfo | Output | Input and output buffer information. See XDM_AlgBufInfo data structure for details. |
| metadataType | XDAS_Int32[] | Input | Type of each metadata plane. |
| decDynamicParams | IVIDDEC3_DynamicParams | Output | Current values of the decoder's dynamic parameters. |

**Note:**

- frameOrder field in the status structure is set to the actual display delay value used by the decoder.

#### 4.2.1.11 IVIDDEC3_OutArgs

**‖ Description**

This structure defines the run-time output arguments for an algorithm instance object.

**‖ Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| size | XDAS_Int32 | Input | Size of the basic or extended (if being used) data structure in bytes. |
| extendedError | XDAS_Int32 | Output | extendedError Field |
| bytesConsumed | XDAS_Int32 | Output | Bytes consumed per decode call |
| OutputID[IVIDEO2_MAX_IO_BUFFERS] | XDAS_Int32 | Output | Output ID corresponding to displayBufs<br>A value of zero (0) indicates an invalid ID. The first zero entry in array will indicate end of valid outputIDs within the array. Hence, the application can stop reading the array when it encounters the first zero entry. |
| decodedBufs | IVIDEO2_BufDesc | Output | The decoder fills this structure with buffer pointers to the decoded frame. Related information fields for the decoded frame are also populated.<br>When frame decoding is not complete, as indicated by outBufsInUseFlag, the frame data in this structure will be incomplete. However, the algorithm will provide incomplete decoded frame data in case application may choose to use it for error recovery purposes. |
| freeBufID[IVIDEO2_MAX_IO_BUFFERS] | XDAS_Int32 | Output | This is an array of inputIDs corresponding to the frames that have been unlocked in the current process call. |
| outBufsInUseFlag | XDAS_Int32 | Output | Flag to indicate that the outBufs provided with the process() call are in use. No outBufs are required to be supplied with the next process() call. |
| displayBufsMode | XDAS_Int32 | Output | Indicates which mode the displayBufs are presented in. See the note below for details. |
| bufDesc [1] | IVIDEO2_BufDesc | Output | Array containing display frames corresponding to valid ID entries in the outputID array.<br>See IVIDEO2_BufDesc data structure for more details. |
| *pBufDesc[IVIDEO2_MAX_IO_BUFFERS] | IVIDEO2_BufDesc * | Output | Array containing pointers to display frames corresponding to valid ID entries in the @c outputID[] |

**Note:**

The display buffer mode can be set as either IVIDDEC3_DISPLAYBUFS_EMBEDDED or IVIDDEC3_DISPLAYBUFS_PTRS.

The current implementation of the decoder will always return a maximum of one display buffer per process call. If the mode is `IVIDDEC3_DISPLAYBUFS_EMBEDDED`, then the instance of the display buffer structure will be present in `OutArgs`. If the mode is `IVIDDEC3_DISPLAYBUFS_PTRS`, then a pointer to the instance will be present in `OutArgs`.

### 4.2.2 MPEG2 Decoder Data Structures

This section includes the following MPEG2 Decoder specific data structures:

- IMPEG2VDEC_Params
- IMPEG2VDEC_DynamicParams
- IMPEG2VDEC_InArgs
- IMPEG2VDEC_Status
- IMPEG2VDEC_OutArgs

#### 4.2.2.1 IMPEG2VDEC _Params

**‖ Description**

This structure defines the creation parameters and any other implementation specific parameters for an MPEG2 Decoder instance object. The creation parameters are defined in the XDM data structure, IVIDDEC3_Params.

**‖ Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| viddec3Params | IVIDDEC3_Params | Input | See IVIDDEC3_Params data structure for details. |
| ErrorConcealmentON | XDAS_Int32 | Input | Set it to 1 to enable error concealment and 0 to disable error concealment.<br><br>If the user wishes to enable error concealment, decoder needs to be operated in IVIDEO_TRANSCODE_FRAMELEVEL mode and application should provide a meta data buffer through process call. metadataType for this buffer should be set to IVIDEO_METADATAPLANE_MBINFO.<br><br>Default value is 0. |
| outloopDeBlocking | XDAS_Int32 | Input | Enables/disables out-of-loop deblocking. Also specifies the strength of deblocking if enabled.<br>0-> Disable de-blocking<br>1-> Enable de-blocking : Minor<br>2-> Enable deblocking : Medium<br>3-> Enable de-blocking : High<br>4-> Enable deblocking : Strong<br>Default value is 0. |

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| debugTraceLevel | XDAS_UInt32 | Input | Specifies the debug trace level. MPEG-2 Decoder supports till level 4. Each higher level logs more debug trace data. Default value is 0 (no debug trace log). |
| lastNFramesToLog | XDAS_UInt32 | Input | Specifies the number of most recent frames to log in debug trace. Minimum value supported is 0 and maximum value supported is 10. Default value is 0. |

#### *4.2.2.2 IMPEG2VDEC_DynamicParams*

**Description**

This structure defines the run-time parameters and any other implementation specific parameters for an MPEG-2 instance object. The run-time parameters are defined in the XDM data structure, IVIDDEC3_DynamicParams.

**Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| viddec3DynamicParams | IVIDDEC3_DynamicParams | Input | See IVIDDEC3_DynamicParams data structure for details. |
| gotoNextIFrame | XDAS_Int32 | Input | If set to 1, only I frames are decoded and provided to the application for display. P/B frames are skipped. This continues until the flag is reset. Default value is 0. |
| skipBFrame | XDAS_Int32 | Input | On setting this flag, decoder skips B frames encountered. It searches for the valid I/P frames and decodes them. This continues until the flag is reset. Default value is 0. |
| skipCurrFrame | XDAS_Int32 | Input | If the flag is set, the current frame is skipped and decoding is continued from the next frame. Default value is 0. |

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| seekFrameEnd | XDAS_Int32 | Input | "seekFrameEnd" is always used in conjunction with flags "gotoNextIFrame", "skipBFrame" or "skipCurrFrame". If this flag is set to 1, decoder seeks to the frame end in case of P/B pictures without decoding and updates the bytes consumed accordingly. ". If this flag is set to 0, decoder just parses the picture type parameter and if the picture type is P/B picture, the decoder returns with bytes consumed set to zero. Default value is 1. |

#### *4.2.2.3 IMPEG2VDEC_InArgs*

**Description**

This structure defines the run-time input arguments for an MPEG2 instance object.

**Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| viddec3InArgs | IVIDDEC3_InArgs | Input | See IVIDDEC3_InArgs data structure for details. |

#### *4.2.2.4 IMPEG2VDEC_Status*

**Description**

This structure defines parameters that describe the status of the MPEG2 Decoder and any other implementation specific parameters. The status parameters are defined in the XDM data structure, IVIDDEC3_Status.

**Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| viddec3Status | IVIDDEC3_Status | Output | See IVIDDEC3_Status data structure for details |
| extendedErrorCode0 | XDAS_UInt32 | Output | Parameter added to capture specific errors not captured in base Status structure |
| extendedErrorCode1 | XDAS_UInt32 | Output | Parameter added to capture specific errors not captured in base Status structure |
| extendedErrorCode2 | XDAS_UInt32 | Output | Parameter added to capture specific errors not captured in base Status structure |

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| extendedErrorCode3 | XDAS_UInt32 | Output | Parameter added to capture specific errors not captured in base Status structure |
| debugTraceLevel | XDAS_UInt32 | Output | Specifies the debug trace level. MPEG2 Decoder supports till level 4. Each higher level logs more debug trace data. |
| lastNFramesToLog | XDAS_UInt32 | Output | Specifies the number of most recent frames to log in debug trace. |
| extMemoryDebugTraceAddr | XDAS_UInt32 * | Output | Specifies the address of the debug trace dump in external memory. |
| extMemoryDebugTraceSize | XDAS_UInt32 | Output | Specifies the size of the debug trace dump in external memory. |

#### *4.2.2.5 IMPEG2VDEC_OutArgs*

**‖ Description**

This structure defines the run-time output arguments for the MPEG2 Decoder instance object.

**‖ Fields**

| Field | Data Type | Input/ Output | Description |
|---|---|---|---|
| viddec3OutArgs | IVIDDEC3_OutArgs | Output | See IVIDDEC3_OutArgs data structure for details. |

## 4.3 Interface Functions

This section describes the application programming interfaces used in the MPEG2 Decoder. The MPEG2 Decoder APIs are logically grouped into the following categories:

- **Creation** – `algNumAlloc()`, `algAlloc()`
- **Initialization –** `algInit()`
- **Control** – `control()`
- **Data processing** – `algActivate()`, `process()`, `algDeactivate()`
- **Termination** – `algFree()`

You must call these APIs in the following sequence:

1) `algNumAlloc()`
2) `algAlloc()`
3) `algInit()`
4) `algActivate()`
5) `process()`
6) `algDeactivate()`
7) `algFree()`

`control()` can be called any time after calling the `algInit()` API.

### 4.3.1 Creation APIs

Creation APIs are used to create an instance of the component. The term creation could mean allocating system resources, typically memory.

**‖ Name**

`algNumAlloc()` – determine the number of buffers that an algorithm requires

**‖ Synopsis**

```
XDAS_Int32 algNumAlloc(Void);
```

**‖ Arguments**

```
Void
```

**‖ Return Value**

```
XDAS_Int32; /* number of buffers required */
```

**‖ Description**

`algNumAlloc()` returns the number of buffers that the `algAlloc()` method requires. This operation allows you to allocate sufficient space to call the `algAlloc()` method.

`algNumAlloc()` may be called at any time and can be called repeatedly without any side effects. It always returns the same result. The `algNumAlloc()` API is optional.

For more details, see TMS320 DSP Algorithm Standard API Reference.

**‖ See Also**

`algAlloc()`

**‖ Name**

`algAlloc()` – determine the attributes of all buffers that an algorithm requires

**‖ Synopsis**

```
XDAS_Int32 algAlloc(const IALG_Params *params, IALG_Fxns
**parentFxns, IALG_MemRec memTab[]);
```

**‖ Arguments**

```
IALG_Params *params; /* algorithm specific attributes */

IALG_Fxns **parentFxns;/* output parent algorithm functions
*/

IALG_MemRec memTab[]; /* output array of memory records */
```

**‖ Return Value**

```
XDAS_Int32 /* number of buffers required */
```

**‖ Description**

`algAlloc()` returns a table of memory records that describe the size, alignment, type, and memory space of all buffers required by an algorithm. If successful, this function returns a positive non-zero value indicating the number of records initialized.

The first argument to `algAlloc()` is a pointer to a structure that defines the creation parameters. This pointer may be `NULL`; however, in this case, `algAlloc()`, must assume default creation parameters and must not fail.

The second argument to `algAlloc()` is an output parameter. `algAlloc()` may return a pointer to its parent's IALG functions. Since the client does not require a parent object to be created, this pointer must be set to `NULL`.

The third argument is a pointer to a memory space of size `nbufs * sizeof(IALG_MemRec)` where, `nbufs` is the number of buffers returned by `algNumAlloc()` and `IALG_MemRec` is the buffer-descriptor structure defined in ialg.h.

After calling this function, `memTab[]` is filled up with the memory requirements of an algorithm.

For more details, see TMS320 DSP Algorithm Standard API Reference.

**Note:**

If you are using extended data structures, the first argument must be a pointer to the extended `Params` data structure. Also, ensure that the `size` field is set to the size of the extended data structure. Depending on the value set for the `size` field, the algorithm uses either base or extended parameters.

**‖ See Also**

`algNumAlloc(), algFree()`

### 4.3.2 Initialization API

Initialization API is used to initialize an instance of the MPEG2 Decoder. The initialization parameters are defined in the `IVIDDEC3_Params` structure (see Data Structures section for details).

**‖ Name**

`algInit()` – initialize an algorithm instance

**‖ Synopsis**

```
XDAS_Int32 algInit(IALG_Handle handle, IALG_MemRec
memTab[], IALG_Handle parent, IALG_Params *params);
```

**‖ Arguments**

```
IALG_Handle handle; /* handle to the algorithm instance*/

IALG_memRec memTab[]; /* array of allocated buffers */

IALG_Handle parent; /* handle to the parent instance */

IALG_Params *params;/*algorithm initialization parameters*/
```

**‖ Return Value**

```
IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */
```

**‖ Description**

`algInit()` performs all initialization necessary to complete the run-time creation of an algorithm instance object. After a successful return from `algInit()`, the instance object is ready to be used to process data.

The first argument to `algInit()` is a handle to an algorithm instance. This value is initialized to the base field of `memTab[0]`.

The second argument is a table of memory records that describe the base address, size, alignment, type, and memory space of all buffers allocated for an algorithm instance. The number of initialized records is identical to the number returned by a prior call to `algAlloc()`.

The third argument is a handle to the parent instance object. If there is no parent object, this parameter must be set to `NULL`.

The last argument is a pointer to a structure that defines the algorithm initialization parameters. All fields in the `params` structure must be set as described in `IALG_Params` structure (see Data Structures section for details).

For more details, see TMS320 DSP Algorithm Standard API Reference.

**Note:**

If you are using extended data structures, the fourth argument must be a pointer to the extended Params data structure. Also, ensure that the size field is set to the size of the extended data structure. Depending on the value set for the size field, the algorithm uses either base or extended parameters.

**‖ See Also**

```
algAlloc(), algMoved()
```

### 4.3.3 Control API

Control API is used for controlling the functioning of MPEG2 Decoder during run-time. This is done by changing the status of the controllable parameters of the decoder during run-time. These controllable parameters are defined in the `IVIDDEC3_DynamicParams` data structure (see Data Structures section for details).

**‖ Name**

`control()` – change run-time parameters of the MPEG2 Decoder and query the decoder status

**‖ Synopsis**

```
XDAS_Int32 (*control)(IVIDDEC3_Handle handle, IVIDDEC3_Cmd
id,IVIDDEC3_DynamicParams *params, IVIDDEC3_Status
*status);
```

**‖ Arguments**

```
IVIDDEC3_Handle handle; /* handle to the MPEG2 decoder
instance */

IVIDDEC3_Cmd id; /* MPEG2 decoder specific control
commands*/

IVIDDEC3_DynamicParams *params /* MPEG2 decoder run-time
parameters */

IVIDDEC3_Status *status /* MPEG2 decoder instance status
parameters */
```

**‖ Return Value**

```
IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */
```

**‖ Description**

This function changes the run-time parameters of MPEG2 Decoder and queries the status of decoder. `control()` must only be called after a successful call to `algInit()` and must never be called after a call to `algFree()`.

The first argument to `control()` is a handle to the MPEG2 Decoder instance object.

The second argument is a command ID. See `IVIDDEC3_Cmd` in enumeration table for details.

The third and fourth arguments are pointers to the `IVIDDEC3_DynamicParams` and `IVIDDEC3_Status` data structures respectively.

**Note:**

If you are using extended data structures, the third argument must be a pointer to the extended `DynamicParams` data structure. Also, ensure that the `size` field is set to the size of the extended data structure. Depending on the value set for the `size` field, the algorithm uses either base or extended parameters.

**‖ See Also**

```
algInit()
```

### 4.3.4 Data Processing API

Data processing API is used for processing the input data using the MPEG2 Decoder.

**‖ Name**

`algActivate()`– initialize scratch memory buffers prior to processing.

**‖ Synopsis**

```
Void algActivate(IALG_Handle handle);
```

**‖ Arguments**

```
IALG_Handle handle; /* algorithm instance handle */
```

**‖ Return Value**

```
Void
```

**‖ Description**

`algActivate()` initializes any of the instance's scratch buffers using the persistent memory that is part of the algorithm's instance object.

The first (and only) argument to `algActivate()` is an algorithm instance handle. This handle is used by the algorithm to identify various buffers that must be initialized prior to calling any of the algorithm's processing methods.

For more details, see *TMS320 DSP Algorithm Standard API Reference*. (literature number SPRU360).

**‖ See Also**

`algDeactivate()`

**‖ Name**

`process()` – basic video decoding call

**‖ Synopsis**

```
XDAS_Int32 (*process)(IVIDDEC3_Handle handle, XDM2_BufDesc
*inBufs, XDM2_BufDesc *outBufs, IVIDDEC3_InArgs *inargs,
IVIDDEC3_OutArgs *outargs);
```

**‖ Arguments**

```
IVIDDEC3_Handle handle; /* handle to the MPEG2 decoder
instance */

XDM2_BufDesc *inBufs; /* pointer to input buffer descriptor
data structure */

XDM2_BufDesc *outBufs; /* pointer to output buffer
descriptor data structure */

IVIDDEC3_InArgs *inargs /* pointer to the MPEG2 decoder
runtime input arguments data structure */

IVIDDEC3_OutArgs *outargs /* pointer to the MPEG2 decoder
runtime output arguments data structure */
```

**‖ Return Value**

```
IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */
```

**‖ Description**

This function does the basic MPEG2 video decoding. The first argument to `process()` is a handle to the MPEG2 Decoder instance object.

The second and third arguments are pointers to the input and output buffer descriptor data structures respectively (see `XDM1_BufDesc` and `XDM_BufDesc` data structure for details).

The fourth argument is a pointer to the `IVIDDEC3_InArgs` data structure that defines the run-time input arguments for the MPEG2 Decoder instance object.

> **Note:**
>
> Prior to each decode call, ensure that all fields are set as described in `XDM2_BufDesc` and `IVIDDEC3_InArgs` structures.

The last argument is a pointer to the `IVIDDEC3_OutArgs` data structure that defines the run-time output arguments for the MPEG2 Decoder instance object.

The algorithm may also modify the output buffer pointers. The return value is `IALG_EOK` for success or `IALG_EFAIL` in case of failure. The `extendedError` field of the `IVIDDEC3_Status` structure contains error conditions flagged by the algorithm. This structure can be populated by calling Control API using `XDM_GETSTATUS` command.

**Note:**

If you are using extended data structures, the fourth argument must be a pointer to the extended `InArgs` data structure respectively. Also, ensure that the `size` field is set to the size of the extended data structure. Depending on the value set for the `size` field, the algorithm uses either base or extended parameters.

**See Also**

`control()`

**‖ Name**

`algDeactivate()` – save all persistent data to non-scratch memory

**‖ Synopsis**

```
Void algDeactivate(IALG_Handle handle);
```

**‖ Arguments**

```
IALG_Handle handle; /* algorithm instance handle */
```

**‖ Return Value**

```
Void
```

**‖ Description**

`algDeactivate()` saves any persistent information to non-scratch buffers using the persistent memory that is part of the algorithm's instance object.

The first (and only) argument to `algDeactivate()` is an algorithm instance handle. This handle is used by the algorithm to identify various buffers that must be saved prior to next cycle of `algActivate()` and processing.

For more details, see TMS320 DSP Algorithm Standard API Reference.

**‖ See Also**

`algActivate()`

### 4.3.5 Termination API

Termination API is used to terminate the MPEG2 Decoder and free up the memory space that it uses.

**‖ Name**

`algFree()` – determine the addresses of all memory buffers used by the algorithm

**‖ Synopsis**

```
XDAS_Int32 algFree(IALG_Handle handle, IALG_MemRec
memTab[]);
```

**‖ Arguments**

```
IALG_Handle handle; /* handle to the algorithm instance */

IALG_MemRec memTab[]; /* output array of memory records */
```

**‖ Return Value**

```
XDAS_Int32; /* Number of buffers used by the algorithm */
```

**‖ Description**

`algFree()` determines the addresses of all memory buffers used by the algorithm. The primary aim of doing so is to free up these memory regions after closing an instance of the algorithm.

The first argument to `algFree()` is a handle to the algorithm instance.

The second argument is a table of memory records that describe the base address, size, alignment, type, and memory space of all buffers previously allocated for the algorithm instance.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

**‖ See Also**

`algAlloc()`

Chapter 5

# Frequenty Asked Questions

This section answers frequently asked questions related to using MPEG-2 Main Profile Decoder on HDVICP2 and Media Controller Based Platform.

## 5.1 Code Build and Execution

| Question | Answer |
|---|---|
| Build error saying that code memory section is not sufficient | Make sure that project settings are not changed from the released package settings such as making project settings as File -O0 and full symbolic debug which throws an error that code memory section is not sufficient. |
| Application returns an error saying “Couldn't open parameter file …..” while running the host test app | Make sure that input file path is given correctly. If the application is accessing input from network, ensure that the network connectivity is stable. |

## 5.2 Issues with Tools Version

| Question | Answer |
|---|---|
| Which tools are required to run the stand-alone codec? | To run the codec on stand-alone setup, you need Framework Components, Code Composer Studio, ARM compiler tools (CG tools).<br>If you are running on the simulator, then the correct version of the HDVICP2 Simulation CSP is needed (See Section 2.1 for more details). |
| What CG tools version should I use for code compilation? | You may use CG tools version 4.5.1 to compile the code. |

## 5.3 Algorithm Related

| Question | Answer |
|---|---|
| Which XDM interface does codec support? | Codec supports XDM IVIDDEC3 interface. |
| Does MPEG-2 Decoder support non-multiple of 16 frame dimensions? | Yes, this decoder supports non-multiple of 16 image dimensions. Even odd resolutions are supported in this version. |
| Does this MPEG-2 Decoder support MPEG-1 constrained video streams? | Yes. |
| Which profiles and levels are supported by this decoder? | This decoder supports Simple Profile upto Main level and Main Profile upto High level. |

| Question | Answer |
|---|---|
| Does this decoder support decoding of interlaced video? | Yes. |
| Does this decoder support “decode header only” feature? | Yes. |
| Does this version of decoder have support for error concealment? | Yes. This version of decoder has support for error concealment (both spatial and temporal). |
| What are the maximum and minimum resolutions supported by the decoder? | This decoder supports resolutions ranging from 64x64 to 2048x2048. |
| Does this version of decoder support display delay? | Only display delay of 1 or 0 frames is supported in this release. Display delay 0 means decoding order. |
| Does Algorithm support DataSync mechanism? | No. This decoder does not support DataSync mechanism for either input or output buffers. |
| Does this decoder support return of MB Info and Error Info to enable transcode applications? | Yes. |
| Does the decoder support meta data output? | No. The decoder does not support returning metadata parsed from the bitstream (except the MBInfo and error info for transcoding, as stated in answer to previous FAQ). |
| Does the decoder support out-of-loop deblocking? | Yes. The decoder supports optional out-of-loop deblocking for display. |
| What is the 128 byte alignment requirement for output buffers? | In case the output buffer is in TILED region, the decoder requires a luma buffer of size ((Width aligned to 16) + 2) * ((Height aligned to 16) + 2) and a chroma buffer of size ((Width aligned to 16) + 2 ) * ((Height/2 aligned to 16) + 2 ). In this case, there is no requirement for a 128 byte aligned picture/output buffer width. Decoder will use the corresponding TILED region pitch as the image pitch.<br><br>In case the output buffer is in non-TILED region, the decoder requires a picture/output buffer with 128 byte aligned width. Thus the buffer requirements will be<br>128_byte_aligned(((Width aligned to 16) + 2)) * ((Height aligned to 16) + 2) for luma buffer and<br>128_byte_aligned(((Width aligned to 16) + 2)) * ((Height/2 aligned to 16) + 2) for chroma buffer<br><br>Please note that the decoder does not assume a 128 byte aligned picture buffer width, so the application has to inform the same using displayWidth feature supported by decoder. Then decoder will use this as image pitch. |

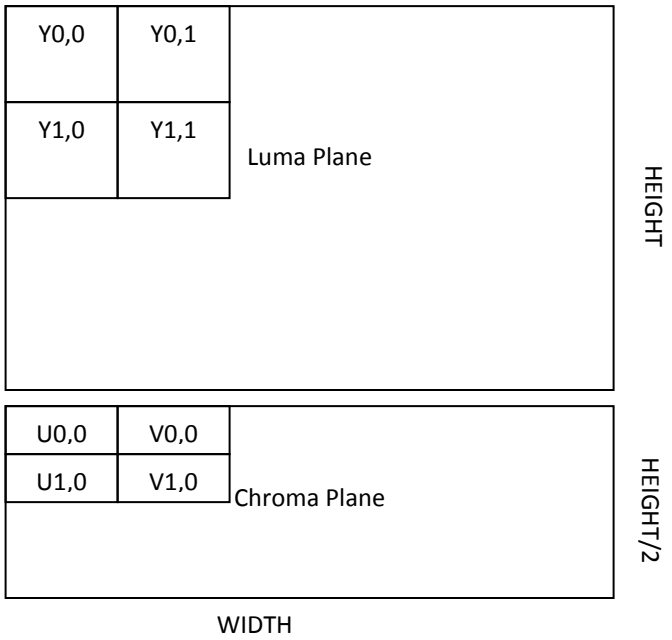
Chapter 6

# Picture Format

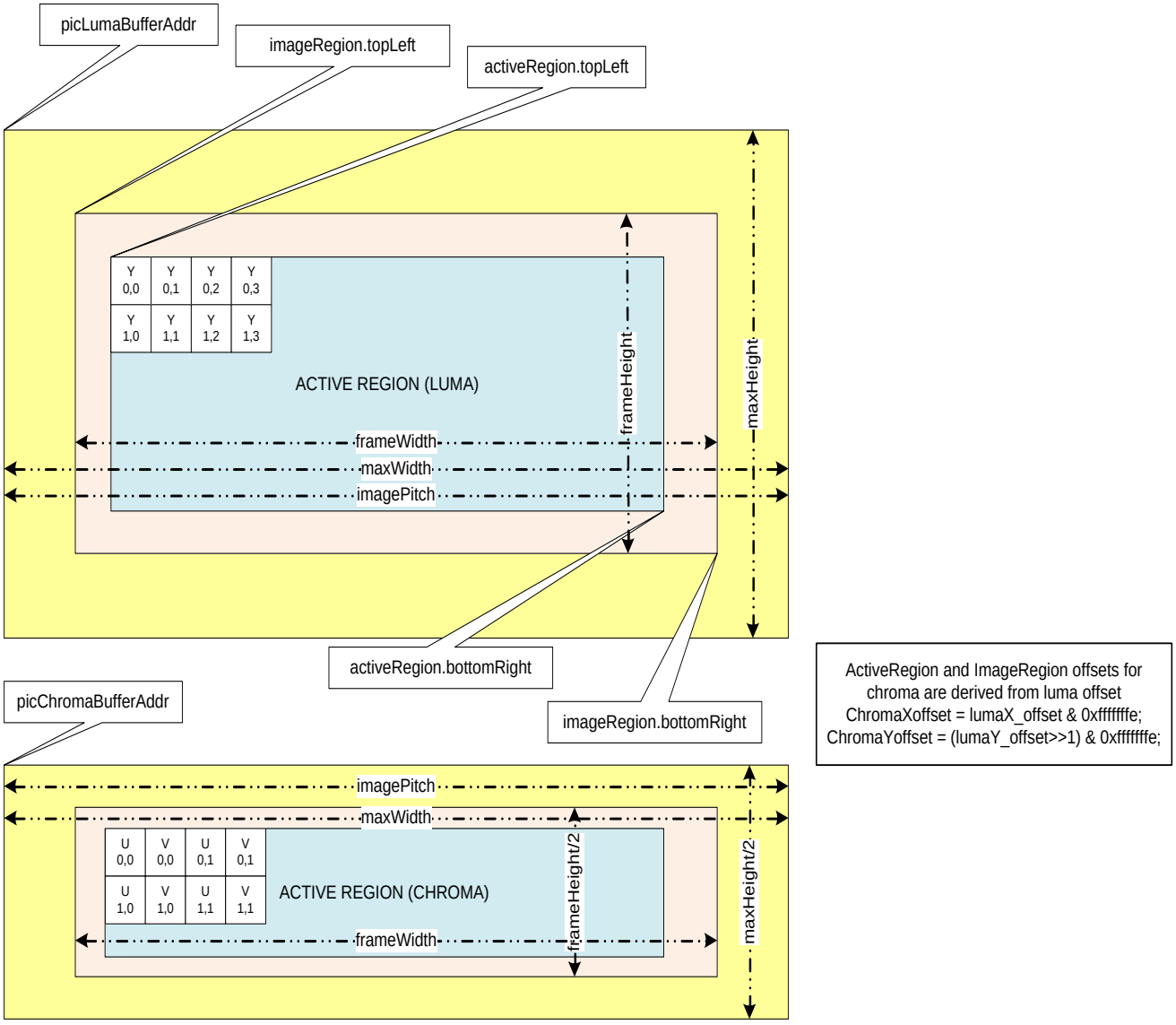
This Appendix explains picture format details for MPEG2 decoder. Decoder outputs YUV frames in NV 12 format.

## 6.1 NV12 Chroma Format

NV12 is YUV 420 semi-planar with two separate planes, one for Y, one for U and V interleaved.



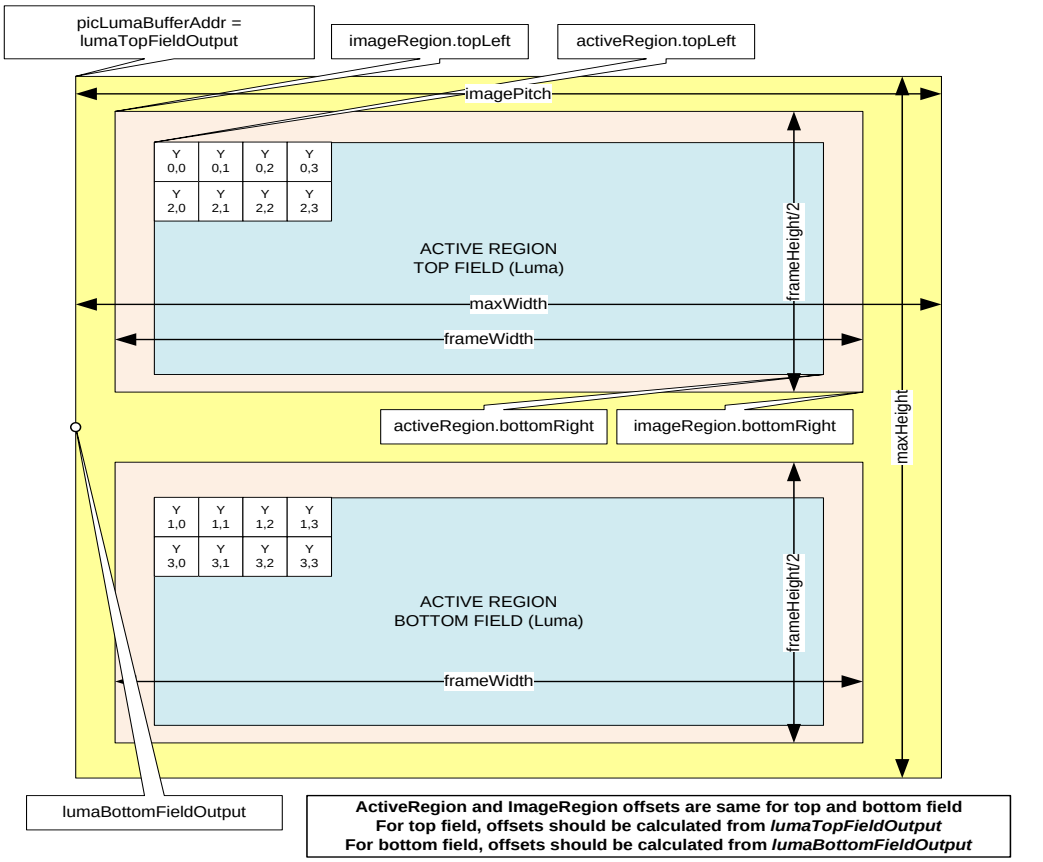
## 6.2 Progressive Picture Format

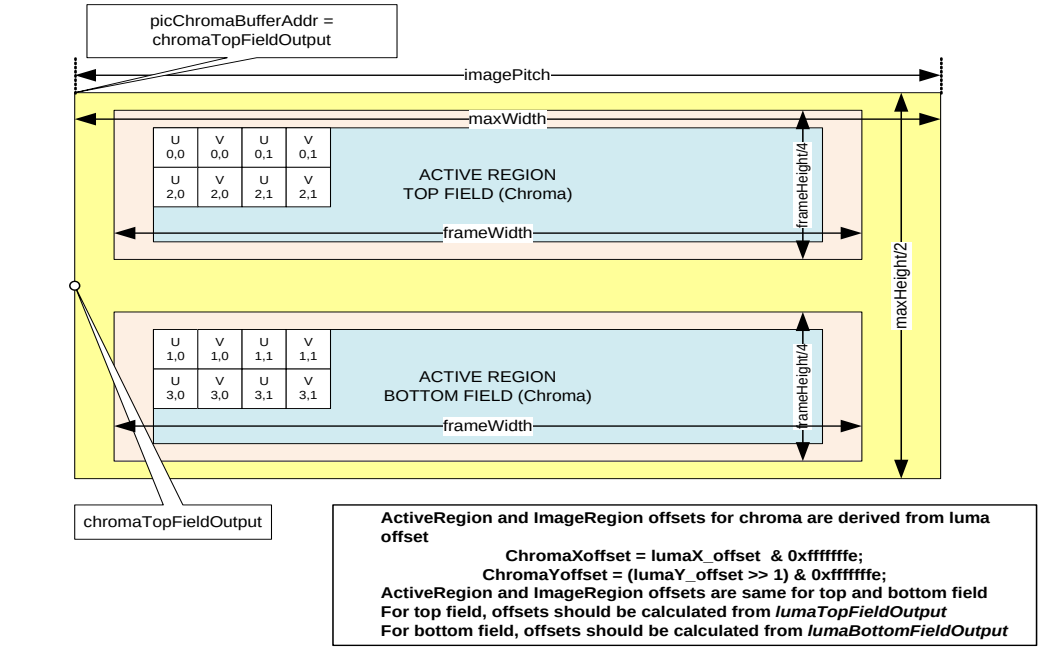


Note that for decoder in case of progressive sequence:

- Luma and chroma buffer addresses can be allocated independently
- Application shall provide this through separate buffer addresses
- The outermost yellow coloured region is the minimum buffer that application should allocate for a given *maxWidth* and *maxHeight*
- *activeRegion*
    - The displayable region
- *imageRegion*
    - Image data decoded by the decoder whose dimensions are always multiple of 16.
    - Contains the activeRegion as a proper subset.
- *Picture Buffer (pic(Luma/Chroma)BufferAddr)*
    - Contains some extra region due to alignment and other constraints.
    - The dimensions (width, height, size) of this buffer should be queried from codec through GETBUFINFO control call
    - Contains the imageRegion as a proper subset.
- *imagePitch*
    - The difference in addresses of two vertically adjacent pixels
    - Typically equal to width of the picture Buffer.
- *Padding Amounts*
    - In vertical direction (bottom), padding amount is 2 pixels for both Luma buffer and chroma buffer.
    - In horizontal direction (right), padding amount is 2 pixels for both Luma buffer chroma buffer.

## 6.3 Interlaced Picture Format



Note that for decoder in case of interlaced sequence:

- Luma and chroma buffers can be allocated independently
- Field buffer allocation cannot be independent
- For every pair of top and bottom field, decoder shall expect a single buffer address from the application
- The outermost yellow coloured region is the minimum buffer that application should allocate for a given *maxWidth* and *maxHeight*
- *activeRegion*
    - The displayable region
- *imageRegion*
    - Image data decoded by the decoder.
    - Contains the activeRegion as a proper subset.
- *Picture Buffer (pic(Luma/Chroma)BufferAddr)*
    - Contains some extra region due to alignment and other constraints.
    - The dimensions (width, height, size) of this buffer should be queried from codec through GETBUFINFO control call
    - Contains the imageRegion as a proper subset.
- *imagePitch*
    - The difference in addresses of two vertically adjacent pixels
    - Typically equal to width of the picture Buffer.
- *Padding Amounts*
    - In vertical direction (bottom), padding amount is 2 pixels for both Luma buffer and chroma buffer.
    - In horizontal direction (right), padding amount is 2 pixels for both Luma buffer chroma buffer.

## 6.4 Constraints on Buffer Allocation for Decoder

- *maxWidth* and *maxHeight* are inputs given to the decoder by the applications
  - Application may not know the output format of the decoder.
  - Therefore, application should allocate Image Buffer based on *maxWidth* and *maxHeight*
    - The extra region beyond the (maxWidth x maxHeight) requirements may be allocated by application due to alignment, pitch or some other constraints
- Application needs to ensure following conditions regarding *imagePitch*
  - *imagePitch* shall be greater or equal to the maxWidth.
  - *imagePitch* shall be multiple of 128 bytes (if the buffer is not in TILED region).
  - *imagePitch* shall actually be the tiler space width (i.e. depends on how many bit per pixel, for 8bpp 16bpp and 32bpp respectively 16Kbyte, 32Kbyte and 32Kbyte). (if the buffer is in TILED region).
  - Application may set *imagePitch* greater than *maxWidth* as per display constraints. However this value must be a multiple of 128 bytes (if the buffer is not in TILED region).
- *picLumaBufferAddr* and *picChromaBufferAddr* shall be 16-byte aligned address. (if the buffer is not in TILED region).
- *ActiveRegion.topLeft* and *ActiveRegion.bottomRight* are decoder outputs
  - Application should calculate actual display width and display height based on these parameters
  - *ActiveRegion.topLeft* and *ActiveRegion.bottomRight* shall be identical for both fields in case of interlaced format
- Maximum and Minimum Resolution is defined as below
  - Progressive
    - Minimum frameWidth = 64
    - Minimum frameHeight = 64
    - Maximum frameWidth = 2048
    - Maximum frameHeight = 2048
  - Interlaced
    - Minimum frameWidth = 64
    - Minimum (frameHeight / 2) = 32
    - Maximum frameWidth = 2048
    - Maximum (frameHeight / 2) = 1024
- Typically picture buffer allocation requirements for decoder, after buffer addresses meet alignment constraints and other requirements, for both progressive and interlaced are as given below.

- Luma buffer size = maxWidth x maxHeight and

  Chroma buffer size = maxWidth x maxHeight/2 where

  - maxWidth = frameWidth + 2 ( additional 2 columns for reference buffer fetch optimization)
  - maxHeight = frameHeight + 2 ( additional 2 rows for reference buffer fetch optimization)

**This page is intentionally left blank**

Chapter 7

# Debug Trace Usage

This section describes the debug trace feature supported by codec and its usage.

## 7.1 Introduction

This section explains the approach and overall design that will be adopted for enabling a trace from a video codec.

The primary uses of Debug Trace are:

1) Make the codec implementation capable of producing a trace containing details about the history of executing a particular instance of the codec

2) Enable the application to dump certain debug parameters from the codec in case of a failure. A failure might even be a hang or crash but in general can be defined as any unacceptable or erroneous behavior

Such a feature is targeted at providing more visibility into the operation of the codec and thus easing and potentially accelerating the process of debug.

## 7.2 Enabling and using debug information

To enable debug information, following two parameters are added to the create time parameters

1) debugTraceLevel

2) lastNFramesToLog

Hence the MPEG-2 decoder create time parameters are modified as

```
typedef struct IMPEG2VDEC_Params{
  IVIDDEC3_Params viddecParams;
  XDAS_Int32     ErrorConcealmentON;
  XDAS_UInt32    outloopDeBlocking;
  XDAS_UInt32    debugTraceLevel;
  XDAS_UInt32    lastNFramesToLog;
  } IMPEG2VDEC_Params;
```

### *7.2.1 debugTracelevel*

This parameter configures the codec to dump a debug trace log

- 0: Disables dumping of debug trace parameters
- >0: Enables the dumping of debug trace parameters. Value specifies the level of debug trace information

### *7.2.2 lastNFramesToLog*

This parameter configures the codec to maintain history of debug trace parameters for last N frames.

- 0: No history will be maintained by the codec
- >0 : History of past specified number of frames will be maintained

In order to avoid book-keeping by the application to know whether the codec has been configured to dump debug trace and where the debug information is available, the following changes are done in the Status structure.

```
typedef struct IMPEG2VDEC_Status{
  IVIDDEC3_Status viddecStatus;
  XDAS_UInt32 extendedErrorCode0;
  XDAS_UInt32 extendedErrorCode1;
  XDAS_UInt32 extendedErrorCode2;
  XDAS_UInt32 extendedErrorCode3;
  XDAS_UInt32 debugTraceLevel;
  XDAS_UInt32 lastNFramesToLog;
  XDAS_UInt32 * extMemoryDebugTraceAddr;
  XDAS_UInt32 extMemoryDebugTraceSize;
} IMPEG2VDEC_Status
```

**debugTraceLevel**: Debug trace level configured for the codec - 0, 1, 2,3,4

**lastNFramesToLog**: Number of frames for which history information is maintained by the codec

**extMemoryDebugTraceAddr**: External memory address (as seen by Media Controller) where debug trace information is being dumped – last memory buffer requested by the codec

**extMemoryDebugTraceSize**: External memory buffer size (in bytes) where debug trace information is being dumped - the size of last memory buffer

Now the application can retrieve this information from the codec at any time by the existing GETSTATUS query through the codec's Control API.

## 7.3 Debug Trace Levels

Debug Debug trace has been (in this implementation) organized into 4 different levels arranged in a hierarchical fashion.

- Level 1 – Frame level information and profile data
- Level 2 – Slice and MB level information
- Level 3 – Logs function call stack for with entry hook
- Level 4 – Logs function call stack for with exit hook

At each higher level, the previous lower levels are also enabled

## 7.4 Requirements On The Application

The following are the requirements on the application side:

1. The application should be capable of configuring *debugTraceLevel* and *lastNFrameToLog* which are part of the Initialization Parameters of the codec
2. The application should be capable of querying the codec for its debug parameter memory regions and size
3. The application should be capable of retrieving these memory regions (In external memory or SL2) for the specified size and preserving these memory dumps in case of any erroneous behavior including a hang/crash.
4. The application, at any time (in case of hang, crash or any unexpected behavior) is expected to be also capable of retrieving the SL2 memory region as returned by the codec in Control-GETSTATUS specified by the SL2 memory debug trace address and size and provide it to the codec developer. The codec developer will have a PC based tool to parse and interpret this dump and produce a readable log of the debug trace parameters.

**This page is intentionally left blank**

Chapter 8

# Error Concealment Usage

This section describes the error concealment feature supported by codec and its usage.

## 8.1 Introduction

When a corrupted input stream is provided to video codec, the codec will decode the stream until the error occurs and comes out of process call gracefully. In this case, the output might have visual artifacts. The amount of artifacts depends on the error type and the amount of valid data before the error occurrence.

Error concealment feature, when enabled will apply concealment algorithms on the improper output and will reduce the amount of distortion in output.

## 8.2 Enabling and using Error concealment

To enable error concealment, following parameter is added to the create time parameters.

- ErrorConcealmentON

Hence the MPEG-2 decoder create time parameters are modified as

```
typedef struct IMPEG2VDEC_Params{
  IVIDDEC3_Params viddecParams;
  XDAS_Int32    ErrorConcealmentON;
  XDAS_UInt32   outloopDeBlocking;
  XDAS_UInt32   debugTraceLevel;
  XDAS_UInt32   lastNFramesToLog;
  } IMPEG2VDEC_Params;
```

Apart from setting the ErrorConcealmentON parameter, user has to enable TRANSCODE mode of operation, set the metadataType and provide an additional metadata buffer for using error concealment feature of MPEG-2 decoder. The same is explained in the following sections.

### *8.2.1 ErrorConcealmentON*

This parameter configures the codec to enabled/disable error concealment.

- 0: Disables concealment
- 1: Enables concealment

### *8.2.2 Transcode Mode*

User has to enable transcode mode of operation to use the concealment feature. This can be achieved by setting the '`operatingMode`' parameter of `IVIDDEC3_Params` to '`IVIDEO_TRANSCODE_FRAMELEVEL`'.

### *8.2.3 MetaData Type*

User has to provide the Metadata type to codec to use concealment feature. This can be achieved by setting the '`metadataType`' parameter of `IVIDDEC3_Params` to `IVIDEO_METADATAPLANE_MBINFO`.

## 8.3 MetaData Buffer Requirement

Once the user sets the codec to work in TRANSCODE mode of operation and `metadataType` during create time, the codec will request for an additional buffer when the application queries for buffer requirement through GETBUFINFO.

The following table compares the codec buffer requirements with and w/o error concealment.

| **Buffer** | **Properties** | **w/o Error concealment** | **With Error concealment** |
|---|---|---|---|
| Number of Input buffers required | | 1 | 1 |
| Inputbuffer | Bit stream | Yes | Yes |
| Number of output buffers required | | 2 | 3 |
| Outputbuffer[0] | Luma buffer | Yes | Yes |
| Outputbuffer[1] | Chroma buffer | Yes | Yes |
| Outputbuffer[2] | Meta data buffer | No | Yes |

Please note that 'Yes' indicates that the corresponding buffer is required and 'No' indicates that the buffer is not required for decoding to work properly.

The required size of the metadata buffer can be calculated from below equation.

*frameSize = (ROUND_TO_NEXT_MULT16(maxWidth) **

*ROUND_TO_NEXT_MULT16(maxHeight))*

*No of MBs in frame = (frameSize) >> 8;*

*Size of the buffer = No of MBs in frame * 112 + frameSize*

The metadata buffer should be a non-TILED buffer.

Following snippet of code explains the example buffer allocation (Refer to sample test application provided).

```
if((MetaDataType == IVIDEO_METADATAPLANE_MBINFO)||(ErrorConcealmentON))
{
 outputBufDesc.numBufs             = 3;
 outputBufDesc.descs[2].bufSize.bytes =
 (XDAS_Int32) buffEle->bufSize[2].bytes;
 outputBufDesc.descs[2].buf        = (XDAS_Int8 *) buffEle->buf[2];
 outputBufDesc.descs[2].memType    =  XDM_MEMTYPE_ROW;
 outputBufDesc.descs[1].accessMask = XDM_ACCESSMODE_READ;
}
```

Codec return the buffer through outArgs->decodebufs->metadataPlaneDesc[0].buf.

## 8.4 Requirements On The Application

The following are the requirements on the application side:

1. The application should be capable of configuring `ErrorconcealmentON`, `operatingMode` and `metadataType` which are part of the Initialization Parameters of the codec.
2. The application should be capable of querying the codec for its memory regions and size using GETBUFINFO.
3. The application should be capable of providing the additional required buffer to codec through process call through outBufsdesc.

# This page is intentionally left blank

# Chapter 9

# Error Handling

This section explains the error handling by MPEG-2 Decoder.

## 9.1 Description

This version of the decoder supports handling of erroneous situations while decoding. If decoder encounters any erroneous situations, it shall exit gracefully without any hang or crash. Also, decoder process call shall return `IVIDDEC3_EFAIL` and relevant error code will be populated in `extendedError` field of `outArgs`.

Some of the erroneous situations will get reported as `XDM_FATALERROR` by the decoder. In certain fatal erroneous situations, the application might flush out the locked buffers, if need be.

Meanings of various error codes and the recommended application behavior are provided in the following tables:

*Table 9-1 Error Codes used to set the extendedError field in IVIDDEC3_OutArgs and IVIDDEC3_Status*

| Bit | Error Code | Explanation | Recommended App Behaviour |
|---|---|---|---|
| 0 | IMPEG2VDEC_ERR_UNSUPPORTED_VIDDEC3PARAMS | This error code has been deprecated. | NA |
| 1 | IMPEG2VDEC_ERR_UNSUPPORTED_VIDDEC3DYNAMICPARAMS | Base class Dynamic params out of supported range | Call GETSTATUS by passing extended Status structure to get more details about the error through extendedErrorCode0 (/1/2/3) parameters |
| 2 | IMPEG2VDEC_ERR_UNSUPPORTED_MPEG2DECDYNAMICPARAMS | Extended class Dynamic params out of supported range | Call GETSTATUS by passing extended Status structure to get more details about the error through extendedErrorCode0 (/1/2/3) parameters |
| 3 | IMPEG2VDEC_ERR_IMPROPER_DATASYNC_SETTING | This error code has been deprecated. | NA |
| 4 | IMPEG2VDEC_ERR_NOSLICE | Illegal start code search error from ECD | The input stream might be corrupted. Pass the next frame in the stream. |
| 5 | IMPEG2VDEC_ERR_SLICEHDR | Illegal Slice start/End error from ECD | The input stream might be corrupted. Pass the next frame in the stream. |
| 6 | IMPEG2VDEC_ERR_MBDATA | Data error detected by ECD | The input stream might be corrupted. Pass the next frame in the stream. |
| 7 | IMPEG2VDEC_ERR_UNSUPPFEATURE | Unsupported header extensions and profile/levels | Call GETSTATUS by passing extended Status structure to get more details about the error through extendedErrorCode0 |

| | | | |
|---|---|---|---|
| | | | (/1/2/3) parameters. Refer Section 1.3 of this user guide for supported services and features. |
| 16 | IMPEG2VDEC_ERR_STREAM_END | End of stream Detected while parsing | Pass the next frame in the stream. |
| 17 | IMPEG2VDEC_ERR_UNSUPPRESOLUTION | Stream resolution out of create time range | Call delete and create again with proper resolutions |
| 18 | IMPEG2VDEC_ERR_STANDBY | HDVICP2 not in standby on acquire | Do HDVICP_Reset, XDM Reset and pass stream |
| 19 | IMPEG2VDEC_ERR_INVALID_MBOX_MESSAGE | Message from iCont to host not defined | Do HDVICP_Reset, XDM Reset and pass stream |
| 20 | IMPEG2VDEC_ERR_HDVICP_RESET | HDVICP2 reset API from host to RMAN failed | Do XDM Reset and pass stream |
| 21 | IMPEG2VDEC_ERR_HDVICP_WAIT_NOT_CLEAN_EXIT | Erroneous exit from the wait API from host to RMAN | Pass the next frame in the stream |
| 22 | IMPEG2VDEC_ERR_SEQHDR | Invalid values of the parameters in sequence header | Pass the next frame in the stream. Decoding will not be proper till the next proper sequence header is encountered. |
| 23 | IMPEG2VDEC_ERR_GOP_PICHDR | Invalid values of the parameters in GOP and picture headers | Pass the next frame in the stream |
| 24 | IMPEG2VDEC_ERR_SEQLVL_EXTN | Invalid values of the parameters in sequence level extension headers | Pass the next frame in the stream. Decoding will not be proper till the next proper sequence header is encountered. |
| 25 | IMPEG2VDEC_ERR_PICLVL_EXTN | Invalid values of the parameters in picture level extension headers | Pass the next frame in the stream |
| 26 | IMPEG2VDEC_ERR_TRICK_MODE | Denotes frame skipped/ reference frame skipped in Trick mode | Pass the next frame in the stream |
| 27 | IMPEG2VDEC_ERR_PICSIZECHANGE | Change in frame dimensions in repeat sequence header | Pass the next frame in the stream |
| 28 | IMPEG2VDEC_ERR_SEMANTIC | Semantic rules of a MPEG-2 stream not followed | Pass the next frame in the stream |
| 29 | IMPEG2VDEC_ERR_DECODE_EXIT | Decoder failed to consume all the input bytes / All MBs not decoded. This has been added to detect the corruption of dimension parameters in header. | Pass the next frame in the stream |
| 30 | IMPEG2VDEC_ERR_IRES_RESHANDLE | Invalid resource handle in IRES interface | Call delete and create again with proper handle |
| 31 | IMPEG2VDEC_ERR_IRES_RESDESC | This error code has been deprecated. | NA |

*Table 9-2 Error Codes used to set the extendedErrorCode0 ,extendedErrorCode1, extendedErrorCode2 and extendedErrorCode3 fields in IMPEG2VDEC_Status*

| Bit | Error Code | Explanation | Recommended App Behaviour |
|---|---|---|---|
| 0 | MPEG2_ECD_ILLEGAL_EOM | ECD3 cannot find EOM, end of macroblock, at the end of macroblock when picture_type is D-picture | The input stream might be corrupted. Pass the next frame in the stream. |
| 1 | MPEG2_ECD_ILLEGAL_EOB | ECD3 cannot find EOB, end of block, in a 64 coefficient block | The input stream might be corrupted. Pass the next frame in the stream. |
| 2 | MPEG2_ECD_ILLEGAL_MP1_ESCAPE_LVL | Decoded level from MPEG-2 ESCAPE code is 0x000 or 0x800 | The input stream might be corrupted. Pass the next frame in the stream. |
| 3 | MPEG2_ECD_ILLEGAL_MP2_ESCAPE_LVL | Decoded level from MPEG-1 ESCAPE code is 0x0000 or 0x8000 | The input stream might be corrupted. Pass the next frame in the stream. |
| 4 | MPEG2_ECD_ILLEGAL_MARKER_CONCEAL | Invalid parameter value in data | The input stream might be corrupted. Pass the next frame in the stream. |
| 5 | MPEG2_ECD_ILLEGAL_MBTYPE_D_PIC | Invalid parameter value in data | The input stream might be corrupted. Pass the next frame in the stream. |
| 6 | MPEG2_ECD_ILLEGAL_DCT_COEFF | Invalid parameter value in data | The input stream might be corrupted. Pass the next frame in the stream. |
| 7 | MPEG2_ECD_ILLEGAL_CBP | Invalid parameter value in data | The input stream might be corrupted. Pass the next frame in the stream. |
| 8 | MPEG2_ECD_ILLEGAL_MOTION_CODE | Invalid parameter value in data | The input stream might be corrupted. Pass the next frame in the stream. |
| 9 | MPEG2_ECD_ILLEGAL_MB_TYPE | Invalid parameter value in data | The input stream might be corrupted. Pass the next frame in the stream. |
| 10 | MPEG2_ECD_ILLEGAL_MB_ADDR_INCR | Invalid parameter value in data | The input stream might be corrupted. Pass the next frame in the stream. |
| 11 | MPEG2_ECD_ILLEGAL_EOS | End of slice cannot be found at the end of picture | The input stream might be corrupted. Pass the next frame in the stream. |
| 12 | MPEG2_ECD_ILLEGAL_QUANT_SCALE_CODE | Decoded quantizer_scale_code is zero | The input stream might be corrupted. Pass the next frame in the stream. |
| 13 | MPEG2_ECD_ILLEGAL_SLICE_START_POS | Showing that two data is mismatched. ECD3 uses macroblock position in ECD3 ON registers and continue processing. | The input stream might be corrupted. Pass the next frame in the stream. |
| 14 | MPEG2_ECD_ILLEGAL_START_CODE_SEARCH | Showing next start code searching infinite error | The input stream might be corrupted. Pass the next frame in the stream. |
| 16 | MPEG2_ECD_ILLEGAL_DC_COEFF_OVFL | Result of DC prediction is overflow or underflow | The input stream might be corrupted. Pass the next frame in the stream. |

| | | | |
|---|---|---|---|
| 17 | MPEG2_DYNAMIC_PARAMS_HANDLE_ERROR | Dynamic parameters handle NULL | Call SETPARAMS with proper dynamicParams |
| 18 | MPEG2_STATUS_HANDLE_ERROR | This error code has been deprecated. | NA |
| 19 | MPEG2_DYNAMIC_PARAMS_SIZE_ERROR | Dynamic params size neither base class or extended class size | Call SETPARAMS with proper values |
| 20 | MPEG2_STATUS_SIZE_ERROR | This error code has been deprecated. | NA |
| 21 | MPEG2_DECODE_HEADER_ERROR | DynamicParams.decodeHeader value out of supported range | Call SETPARAMS with proper values |
| 22 | MPEG2_DISPLAY_WIDTH_ERROR | DynamicParams.displayWidth value out of supported range | Call SETPARAMS with proper values |
| 23 | MPEG2_FRAME_SKIP_MODE_ERROR | DynamicParams.frameSkipMode value out of supported range | Call SETPARAMS with proper values |
| 24 | MPEG2_NEW_FRAME_FLAG_ERROR | DynamicParams.newFrameFlag value out of supported range | Call SETPARAMS with proper values |
| 25 | MPEG2_GOTO_IFRAME_ERROR | Extended Dynamic param gotoNextIFrame out of supported range | Call SETPARAMS with proper values |
| 26 | MPEG2_SKIP_BFRAME_ERROR | Extended Dynamic param skipBFrame out of supported range | Call SETPARAMS with proper values |
| 27 | MPEG2_SKIP_CURRENTFRAME_ERROR | Extended Dynamic param skipCurrFrame out of supported range | Call SETPARAMS with proper values |
| 28 | MPEG2_SEEK_FRAMEEND_ERROR | Extended Dynamic param seekFrameEnd out of supported range | Call SETPARAMS with proper values |
| 29 | MPEG2_NULL_STATUS_DATA_BUF | Data Buffer pointer in status structure NULL in GETVERSION control call | Call GETVERSION with proper data buffer pointer in status structure |
| 30 | MPEG2_INSUFFICIENT_STATUS_DATA_BUF | Data buffers within status structure less than 96 bytes during getversion control call | Call GETVERSION with proper data buffer size |
| 31 | MPEG2_NULL_INARGS_POINTER_ERROR | InArgs pointer in process call NULL | Call process call with valid inArgs pointer to process call |
| 32 | MPEG2_INARGS_SIZE_ERROR | InArgs size in process call neither base class size nor extended class size | Call process call with valid size for inArgs during process call |
| 33 | MPEG2_INVALID_INPUT_BYTES_ERROR | Non-positive value of the input bytes(pInArgs->numBytes) to process call | Call process call with a valid input buffer size |
| 34 | MPEG2_INVALID_INPUT_ID_ERROR | The input ID of output buffer is 0 | Call process call with a valid input ID |
| 35 | MPEG2_DECODER_NOT_INITIALIZED_ERROR | This error code has been deprecated. | NA |
| 36 | MPEG2_NULL_INPUT_BUF_DESC_ERROR | Pointer to the Input Buffer descriptor is NULL | Call process call with valid input buffer descriptor pointer to process call |
| 37 | MPEG2_NULL_INPUT_BUFFER_POINTER_ERROR | Input buffer pointer is NULL | Call process call with valid input buffer pointer to process call |
| 38 | MPEG2_INVALID_INPUT_BUFFER_SIZE_ERROR | Invalid Input buffer size: when non-positive values for bytes in raw and dimensions in tiler | Call process call with valid input buffer size to process call |
| 39 | MPEG2_INVALID_NUM_OF_INPUT_BUFFERS_ERROR | Insufficient buffers (when number of Input Buffers less than 1) | Call process call with sufficient number of input buffers to process call |
| 40 | MPEG2_EXCESS_NUM_OF_INPUT_BUFFERS_ERROR | Redundant input buffers (greater than 1) | Pass the next frame in the stream with appropriate number of input buffers |
| 41 | MPEG2_INVALID_INPUT_BUFFER_MEMTYPE_ERROR | Input stream buffer memory type other than RAW/Page mode | Call process call with valid input buffer memory type to process call |
| 42 | MPEG2_NULL_OUTARGS_POINTER_ERROR | OutArgs pointer is NULL | Call process call with valid outArgs pointer to process call |
| 43 | MPEG2_INVALID_OUTARGS_SIZE | OutArgs size set to value other than base class or extended class size | Call process call with valid size for outArgs during process call |
| 44 | MPEG2_NULL_OUTPUT_BUF_DESC_POINTER_ERROR | The Display buffer descriptor pointer in OutArgs is NULL | Call process call with valid input buffer descriptor pointer to |

| | | | |
|---|---|---|---|
| | | | process call |
| 45 | MPEG2_NULL_OUTPUT_BUF_DESC_ERROR | The Output buffer descriptor pointer provided in process call is NULL | Call process call with valid output buffer descriptor pointer to process call |
| 46 | MPEG2_INVALID_OUTPUT_BUFFER0_POINTER_ERROR | Luma output buffer pointer is NULL | Call process call with valid luma output buffer pointer to process call |
| 47 | MPEG2_INVALID_OUTPUT_BUFFER0_SIZE_ERROR | Luma output buffer size/dimensions being non-positive | Call process call with valid luma output buffer size to process call |
| 48 | MPEG2_INVALID_NUM_OF_OUTPUT_BUFFERS_ERROR | Number of output buffers other than 2 (3 in metadataType Mbinfo) | Call process call with appropriate number of output buffers to process call |
| 49 | MPEG2_INVALID_OUTPUT_BUFFER0_MEMTYPE_ERROR | Luma buffer memory other than Raw, tiled8, and tiledpage mode | Call process call with valid luma output buffer memory type to process call |
| 50 | MPEG2_INVALID_OUTPUT_BUFFER0_ALIGNMENT_ERROR | Luma buffer base address not 128-bit aligned | Call process call with properly aligned luma output buffer to process call |
| 51 | MPEG2_INVALID_OUTPUT_BUFFER1_POINTER_ERROR | Chroma output buffer pointer is NULL | Call process call with valid chroma output buffer pointer to process call |
| 52 | MPEG2_INVALID_OUTPUT_BUFFER1_SIZE_ERROR | Chroma output buffer size/dimensions being non-positive | Call process call with valid chroma output buffer size to process call |
| 53 | MPEG2_INVALID_OUTPUT_BUFFER1_MEMTYPE_ERROR | Chroma buffer memory other than Raw, tiled 8, tiled16 and page mode | Call process call with valid chroma output buffer memory type to process call |
| 54 | MPEG2_INVALID_OUTPUT_BUFFER1_ALIGNMENT_ERROR | Chroma buffer base address not 128-bit aligned | Call process call with properly aligned chroma output buffer to process call |
| 55 | MPEG2_INVALID_OUTPUT_BUFFER2_POINTER_ERROR | MBinfo output buffer pointer is NULL in metadataType Mbinfo mode | Call process call with valid MBInfo output buffer pointer to process call |
| 56 | MPEG2_INVALID_OUTPUT_BUFFER2_SIZE_ERROR | MBinfo output buffer size is non-positive in metadataType Mbinfo mode | Call process call with valid MBInfo output buffer size to process call |
| 57 | MPEG2_INVALID_OUTPUT_BUFFER2_MEMTYPE_ERROR | MBinfo output buffer memory type other than RAW in metadataType Mbinfo mode | Call process call with valid MBInfo output buffer memory type to process call |
| 58 | MPEG2_INVALID_BUFFER_USAGE_MODE | This error code has been deprecated. | NA |
| 59 | MPEG2_SEQ_HDR_INVALID_FRAME_WIDTH | Frame width from header being 0 or greater than max width provided at create time | Pass the next frame in the stream |
| 60 | MPEG2_SEQ_HDR_INVALID_FRAME_HEIGHT | Frame height from header being 0 or greater than max height provided at create time | Pass the next frame in the stream |
| 61 | MPEG2_SEQ_HDR_INVALID_ASPECT_RATIO | Aspect ratio from sequence header is not standard compliant | Pass the next frame in the stream |
| 62 | MPEG2_SEQ_HDR_INVALID_FRAME_RATE_CODE | Frame rate code from sequence header not standard compliant | Pass the next frame in the stream |
| 63 | MPEG2_INVALID_INTRA_QUANT_MAT | Intra quantization matrix in sequence header or quantization matrix extension is not standard compliant | Pass the next frame in the stream |
| 64 | MPEG2_INVALID_NON_INTRA_QUANT_MAT | Non-Intra quantization matrix in sequence header or quantization matrix extension is not standard compliant | Pass the next frame in the stream |
| 65 | MPEG2_SEQ_HDR_INVALID_INTRA_ESCAPE_BIT | Escape bit in ProfileandLevel indication of sequence extension is not standard compliant | Pass the next frame in the stream |
| 66 | MPEG2_SEQ_HDR_INVALID_PROFILE | Unsupported Profile, other than simple and main | Pass the next frame in the stream |
| 67 | MPEG2_SEQ_HDR_INVALID_LEVEL | Invalid level indication in sequence extension | Pass the next frame in the stream |
| 68 | MPEG2_SEQ_HDR_INVALID_RESOLUTION_FORLVL | Level based resolution constraints violated | Pass the next frame in the stream |
| 69 | MPEG2_SEQ_HDR_INVALID_CHROMA_FORMAT | Chroma format in sequence extension other than 4:2:0 | Pass the next frame in the stream |

| | | | |
|---|---|---|---|
| 70 | MPEG2_SEQ_HDR_INVALID_LOW_DELAY | Low-delay flag set to '1' in sequence extension not supported | Pass the next frame in the stream |
| 71 | MPEG2_SEQ_DSP_INVALID_VIDEO_FORMAT | Video format parameter in sequence display extension not standard compliant | Pass the next frame in the stream |
| 72 | MPEG2_SEQ_DSP_INVALID_COLOUR_PRIM | Colour primaries parameter in sequence display extension not standard compliant | Pass the next frame in the stream |
| 73 | MPEG2_SEQ_DSP_INVALID_TRF_CHARS | Transfer characteristics parameter in sequence display extension not standard compliant | Pass the next frame in the stream |
| 74 | MPEG2_SEQ_DSP_INVALID_MAT_COEFFS | Matrix coefficients for colour conversion in sequence display extension not standard compliant | Pass the next frame in the stream |
| 75 | MPEG2_GOP_HDR_INVALID_DROP_FLAG | The drop flag in GOP header set when frame rate is not 29.97 Hz | Pass the next frame in the stream |
| 76 | MPEG2_GOP_HDR_INVALID_HOUR | Invalid 'hour' parameter in GOP header | Pass the next frame in the stream |
| 77 | MPEG2_GOP_HDR_INVALID_MIN | Invalid 'minute' parameter in GOP header | Pass the next frame in the stream |
| 78 | MPEG2_GOP_HDR_INVALID_SEC | Invalid 'second' parameter in GOP header | Pass the next frame in the stream |
| 79 | MPEG2_GOP_HDR_INVALID_TIME_CODE_PICTURES | Invalid 'frame' parameter in GOP header | Pass the next frame in the stream |
| 80 | MPEG2_GOP_HDR_INVALID_BROKEN_LINK | Broken link in GOP header for a closed GOP | Pass the next frame in the stream |
| 81 | MPEG2_PIC_HDR_INVALID_TEMP_REF | This error code has been deprecated. | Pass the next frame in the stream |
| 82 | MPEG2_PIC_HDR_INVALID_PIC_TYPE | Invalid picture type code: D-picture in MPEG2 and B-picture in MPEG1 simple profile. | Pass the next frame in the stream |
| 83 | MPEG2_PIC_HDR_INVALID_VBV_DELAY | This error code has been deprecated. | Pass the next frame in the stream |
| 84 | MPEG1_PIC_HDR_INVALID_FWD_FCODE | Forward f-code in picture header for B/P pictures is '0' | Pass the next frame in the stream |
| 85 | MPEG1_PIC_HDR_INVALID_BWD_FCODE | Backward f-code in picture header for B pictures is '0' | Pass the next frame in the stream |
| 86 | MPEG2_PIC_HDR_INVALID_FCODE | Invalid fcode values in picture extension | Pass the next frame in the stream |
| 87 | MPEG2_PIC_HDR_INVALID_PIC_STRUCTURE | Invalid picture_structure value in picture extension. Also, progressive sequence has to contain 'frame' pictures. | Pass the next frame in the stream |
| 88 | MPEG2_PIC_HDR_INVALID_FIELD_COMB | Picture type combinations for filed pictures not standard compliant. Valid types: II, IP, PP, BB | Pass the next frame in the stream |
| 89 | MPEG2_PIC_HDR_INVALID_TFF | Top_field_first flag in picture coding extension not standard compliant | Pass the next frame in the stream |
| 90 | MPEG2_PIC_HDR_INVALID_FPFD | framePredFrameDct flag in picture coding extension not standard compliant: has to be '1' for progressive sequence; has to be '0' for field pictures | Pass the next frame in the stream |
| 91 | MPEG2_PIC_HDR_INVALID_RFF | RepeatFirstField flag in picture coding extension had to be '0' for a progressive sequence | Pass the next frame in the stream |
| 92 | MPEG2_PIC_HDR_INVALID_PROG_FLAG | Picture structure has to be 'frame' in a progressive frame | Pass the next frame in the stream |
| 93 | MPEG2_QUANT_EXT_INVALID_LOAD_CHROMA_INTRA_FLAG | load_chroma_intra_quantizer_matrix shall be '0' for 4:2:0 format | Pass the next frame in the stream |
| 94 | MPEG2_QUANT_EXT_INVALID_LOAD_CHROMA_NON_INTRA_FLAG | load_chroma_non_intra_quantizer_matrix shall be '0' for 4:2:0 format | Pass the next frame in the stream |
| 95 | MPEG2_INVALID_EXTN_CODE | Undefined header extension code | Refer Section 1.3 of this user guide for supported services and features. |
| 96 | MPEG2_SEQ_HDR_MISSING | Picture header is encountered before sequence header | Pass the next frame in the stream |

| | | | |
|---|---|---|---|
| 97 | MPEG2_NO_PICTURE_ENCODED_ERROR | This error code has been deprecated. | NA |
| 98 | MPEG2_SEQ_EXT_MISSING | Occurrence of picture coding extension without sequence extension | Pass the next frame in the stream |
| 99 | MPEG2_PIC_CODING_EXT_MISSING | This error code has been deprecated. | NA |
| 100 | MPEG2_SEQ_DISP_EXT_MISSING | This error code has been deprecated. | NA |
| 101 | MPEG2_GOP_FIRST_FRAME_NOT_I | First picture of a GOP not an I picture | Pass the next frame in the stream |
| 102 | MPEG2_SCALABILITY_NOT_SUPPORTED | Scalable extensions not supported, hence just parsed | Pass the next frame in the stream |
| 103 | MPEG2_END_OF_SEQ_DETECTED | Sequence end code detected on parsing | End of sequence has been reached |
| 104 | MPEG2_PIC_HDR_RFF_FRAME_RATE_MISMATCH | Standard defined dependency of repeatFirstField flag on frame rate violated | Pass the next frame in the stream |
| 105 | MPEG2_PIC_HDR_INVALID_DC_PRECISION | intraDcPrecision decides the scaling factor in the inverse quantization process for the DC coefficient of an intra block. This has an invalid value. | Pass the next frame in the stream |
| 106 | MPEG2_INVALID_FRAME_RATE | Frame rate from header is '0' or greater than the max value set during create time | Pass the next frame in the stream |
| 107 | MPEG2_INVALID_BIT_RATE | Bit-rate from header is '0' or greater than the max value set during create time | Pass the next frame in the stream |
| 108 | MPEG2_FRAME_SKIPPED | Indicates that the current picture has been skipped from decoding in trick mode | Pass the next frame in the stream |
| 109 | MPEG2_REF_FRAME_SKIPPED | Indicates that the reference picture for P/B piture has been skipped under trick mode | Pass the next frame in the stream |
| 110 | MPEG2_NO_REF_TO_FLUSH | No more reference pictures held by the codec to provide in the flush call | All reference pictures have been given out by the codec |
| 111 | MPEG2_EXCESS_INPUT_BYTES | Entire input bytes have not been consumed by the codec | Pass the next frame in the stream |
| 112 | MPEG2_ALL_MBS_NOT_DECODED | Number of decoded MBs is lesser than the expected number of MBs in the picture | Pass the next frame in the stream |