MJPEG Decoder on HDVICP2 and Media Controller Based Platform

User's Guide

Literature Number: SPRUH44
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) MJPEG Decoder implementation on the HDVICP2 and Media Controller 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 MJPEG 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 codec and its usage.
- **Chapter 8 – Data Sync API Usage,** explains the sub-frame level data synchronization API usage for MJPEG Decoder from application point of view.
- **Chapter 9 – Error Handling,** explains the error handling and error robustness features of this MJPEG Decoder.
- **Chapter 10 – Slice Level Decoding,** describes the slice level decoding feature supported by MJPEG Decoder and its usage.

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 IS 10918-1 Information Technology - Digital Compression and Coding of Continuous-Tone Still Images -- Part 1: Requirements and Guidelines* | *CCITT Recommendation T.81*

Abbreviations

The following abbreviations are used in this document.

List of Abbreviations

Abbreviation	**Description**
BIOS	TI's simple RTOS for DSPs
CSL	Chip Support Library
D1	720x480 or 720x576 resolutions in progressive scan
DCT	Discrete Cosine Transform
DMA	Direct Memory Access
EVM	Evaluation Module
HDTV	High Definition Television
IRES	Interface standard to request and receive handles to resources
ISO	International Standards Organization
IVA	Image Video Accelerator
MCU	Minimum Coded Unit
JPEG	Joint Photographic Experts Group
NTSC	National Television Standards Committee
RMAN	Resource Manager
RTOS	Real Time Operating System
VGA	Video Graphics Array (640 x 480 resolution)
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 (MJPEG 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.

Contents

This page is intentionally left blank

Figures

This page is intentionally left blank

Tables

This page is intentionally left blank

Chapter 1

Introduction

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

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

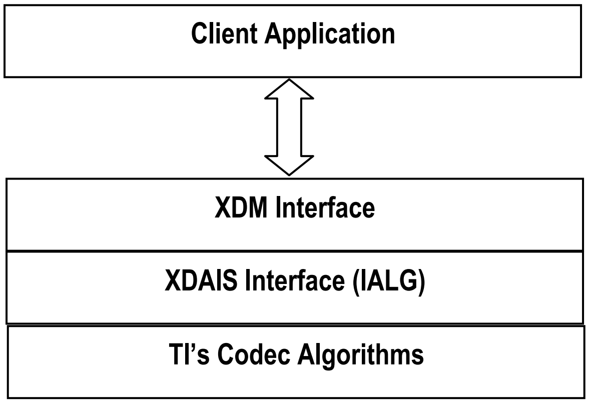
In the multimedia application space, you have the choice of integrating any codec into your multimedia system. For example, if you are building a video decoder system, you can use any of the available video decoders (such as MPEG4, H.263, or MJPEG) in your system. To enable easy integration with the 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).

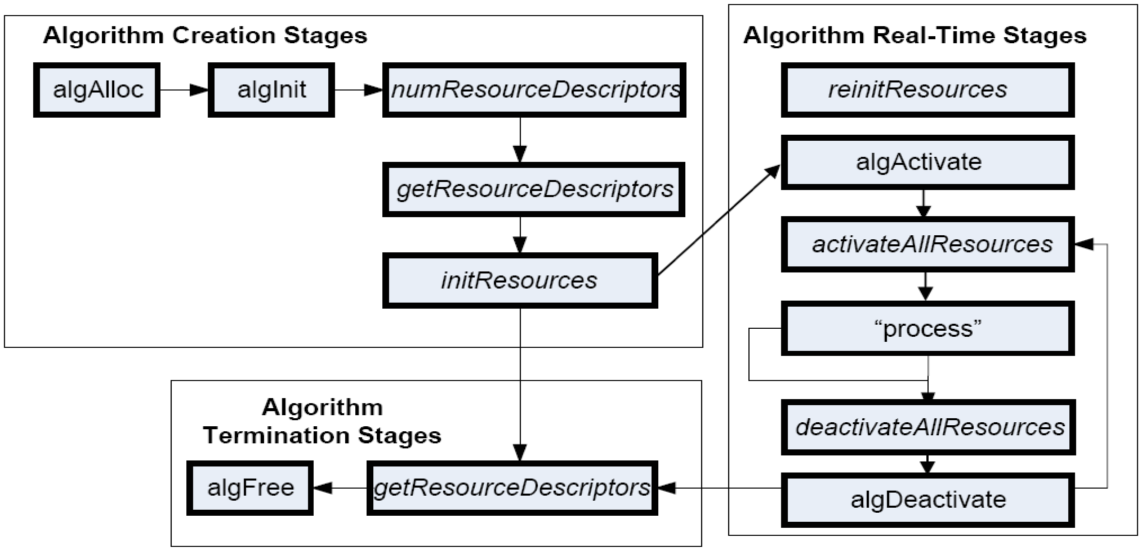


Figure 1-1 IRES Interface Definition and Function Calling Sequence

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

1.2 Overview of MJPEG Decoder

JPEG is an international standard for color image compression. This standard is defined in the ISO 10918-1 JPEG Draft International Standard | CCITT Recommendation T.81. It is a widely used Image compression algorithm that uses Inverse Quantization, Inverse Discrete Cosine Transform (IDCT) coding of the residual data and Huffman entropy coding.

Some important JPEG modes are:

- ❑ Sequential DCT based
- ❑ Progressive DCT based
- ❑ Hierarchical
- ❑ Lossless

Following are the supported processes and features as per JPEG standard:

Baseline:

- 8bit samples per component
- Sequential only
- Huffman coding uses 2 AC and 2 DC tables

Extended:

- 8 or 12 bit samples per component
- Both Sequential and Progressive
- Huffman or Arithmetic coding has 4 AC and 4DC Tables

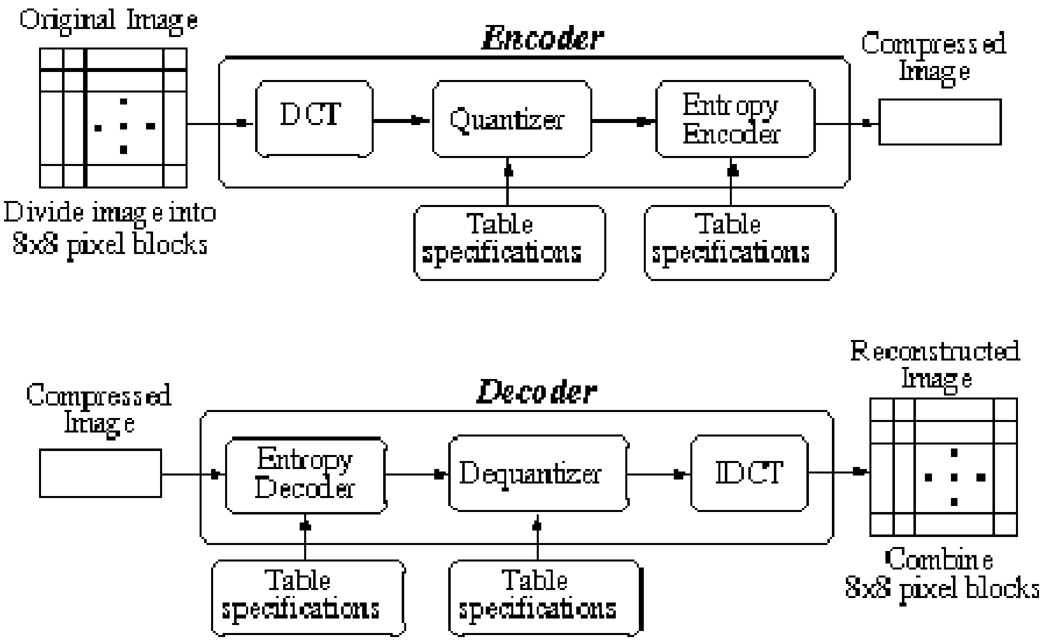


Figure 1-2 Block diagram of Simple JPEG encoder-decoder

From this point onwards, all references to MJPEG Decoder means JPEG Baseline Sequential Decoder used in video mode (i.e. in continuous multiple JPEG image decoding mode).

1.3 Supported Services and Features

This user guide accompanies TI’s implementation of MJPEG Decoder on the HDVICP2 platform.

This version of the codec has the following supported features:

- eXpressDSP Digital Media (XDM IVIDDEC3) compliant
- Supports baseline sequential mode with both interleaved and non-interleaved input formats
- Supports 8bpp per component
- Supports extended sequential mode with some constraints. Does not support arithmetic decoding and 12 bits per sample.
- Supports YUV444, YUV422, YUV420 and YUV400 chroma sub-sampling formats for input
- Both horizontal down sampling and vertical down sampling supported for input YUV422 images i.e. this MJPEG decoder can decode both horizontally downsampled and vertically downsampled YUV422 images

- Supports YUV 444 planar, YUV 422 IBE (YUYV) and YUV 420 semi-planar chroma sub-sampling formats for output. Please refer to Table 1-1.
- Supports a maximum of three components
- Supports all resolutions up to 4096x4096
- Supports 8-bit and 16-bit quantization tables
- Supports a maximum of four Huffman tables each for AC and DC DCT coefficients
- Supports decoding of custom Huffman tables
- Supports decoding of JPEG File Interchange Format (JFIF) header
- Supports parsing of Comment marker
- Supports decoding of EXIF marker
- Supports parsing of restart marker
- Skips all unsupported markers
- Supports sub-frame data synchronization for input and output
- Supports graceful exit under error conditions
- Supports multi-channel functionality
- Supports thumbnail for preview. The thumbnail can be JFIF or EXIF. Thumbnail can be RGB as well as JPG. The user can also specify that a downsampled version of the image be given out as thumbnail.
- Supports scaling for YUV444 and YUV400 images
- Supports error concealment (for YUV420 interleaved input only)
- Supports slice level decoding through user configurable parameters
- Supports debug trace dump

Limitations:

- Does not support arithmetic decoding
- Does not support 12 bits per sample
- Does not support return of metadata present in JFIF, Exif and comment markers to application (but supports decoding of thumbnail images embedded in JFIF or Exif markers)
- Does not support post-processing algorithms such as (a) Rotation (b) Scaling with arbitrary ratio (c) Flipping (d) out of loop de-blocking filter (e) out loop de-ring filter (f) chroma conversion (g) alpha blending
- Does not support slice level switching and sub frame level data synchronization simultaneously. Their support is mutually exclusive.

Table 1-1 Chroma Formats Supported

Input Image Format	**Output chroma formats supported**
420 Interleaved	YUV420 Semi Planar
422 (Horizontally downsampled) Interleaved	YUV422 IBE (YUYV)(1), YUV420 Semi Planar
422 (Vertically downsampled) Interleaved	YUV422 Planar
444 Interleaved	YUV444 Planar, YUV420 Semi Planar(1)
400 (Grayscale)	YUV420 Semi Planar with chroma set to 0x80
Non-interleaved (multiple scan) images	The output will be planar with the same chroma subsampling format as input

(1) These output formats require conversion by software. So, significant performance deviation would be seen for these output formats as compared to the other supported output formats.

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 has been 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 codec has been compiled, assembled, archived, and linked using the code generation tools version 4.5.1.

  All though 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

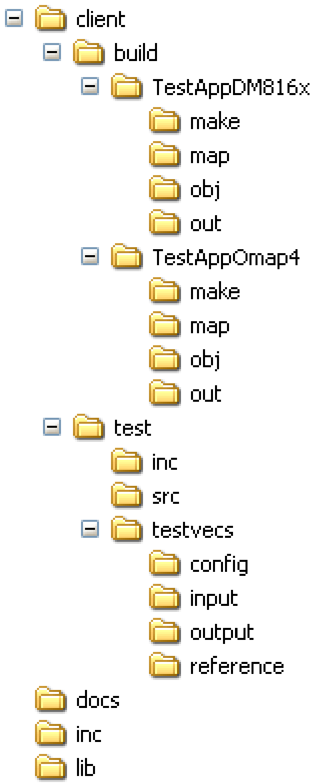
  This codec has also been tested using Netra CSP (Simulation) version 0.7.1. This version of Simulator can be downloaded through software updates on Code Composer Studio v4. Ensure that the following site is listed as part of “Update sites to visit”.

  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 directory called 500.V.MJPEG.D.IVAHD.01.00 under which under which the directory named IVAHD_001 is created:

The sub directory structures for IVAHD_001 are depicted in Figure 2-1.



```
⊟ client
    ⊟ build
        ⊟ TestAppDM816x
            make
            map
            obj
            out
        ⊟ TestAppOmap4
            make
            map
            obj
            out
    ⊟ test
        inc
        src
        ⊟ testvecs
            config
            input
            output
            reference
docs
inc
lib
```

Figure 2-1. Component Directory Structure

Table 2-1 provides a description of the sub-directories created in the 500.V.MJPEG.D.IVAHD.01.00/IVAHD_001 directory.

Table 2-1 Component Directories

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

Sub-Directory	Description
\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
\client\test\testvecs\config	Contains sample configuration file for MJPEG 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 MJPEG Decoder
\lib	Contains jpegvdec_ti_host.lib – HDVICP2 MJPEG Decoder built as a library on Media Controller

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 Iivahd_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 <CCSv4.2_InstallFolder>\ccsv4\tools\compiler\tms470.

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_HDVICP2 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. This decoder has been validated with XDC version 3.20.04.68 GA. 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

This library release of MJPEG Decoder on HDVICP2 and Media Controller based platform contains the following projects.

Project	Make file Path	Output files
Test Application	`\client\build\<TestAppDeviceName>\make\`	`\client\build\TestApp<DeviceName>\out\jpegvdec_ti_testapp.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.MJPEG.D.IVAHD.01.00\IVAHD_001\client\build\TestAppDeviceName\out\ sub-directory, select the codec executable

"jpegvdec_ti_hosttestapp.out" and load it into Code Composer Studio in preparation for execution.

5) Select IVAHD_0_ICONT1 device and **Target > Run** to give iCont1 device a free run.

6) Select IVAHD_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.MJPEG.D.IVAHD.01.00\IVAHD_001\client\test\testvecs\input\ directory and generates outputs in \500.V.MJPEG.D.IVAHD.01.00\IVAHD_001\client\test\testvecs\output\ directory.

2.4.3 Running the Sample Test Application on DM816x EVM

To run the sample test application on DM816x DDR2 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 configuration 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 CortexM3.

5) Select Cortex_M3_RTOS_0 device and **Target > Load Program**, browse to \500.V.MJPEG.D.IVAHD.01.00\IVAHD_001\client\build\TestAppDM816x\out \ sub-directory, select the codec executable "jpegvdec_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.MJPEG.D.IVAHD.01.00\IVAHD_001\client\test\testvecs\input\ directory and generates outputs in \500.V.MJPEG.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:

```
Mode
Config
Input
Output
```

where:

- `Mode` 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

A sample Testvecs.cfg file is as shown:

```
0
..\..\Test\TestVecs\Config\Testparams.cfg
..\..\Test\TestVecs\Input\davincieffect_qcif_yuv420_5fr.mjpg
..\..\Test\TestVecs\Output\davincieffect_qcif_yuv420_5fr.yuv
```

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. Compliance mode has not been implemented in this release of JPEG Decoder.

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:

```
# <ParameterName> = <ParameterValue> # Comment
############################################################
# Parameters
############################################################

ImageWidth              = 4096 # Max image width in Pels

ImageHeight             = 4096 # Max image height in Pels

ChromaFormat            = 9    # Output Chroma Format
                               # 9=>YUV420SP
                               # 5=>YUV444P
                               # 3=>YUV422 YUYV

FramesToDecode          = 1 # Number of frames to be decoded

DumpFrom                = 0 # Start dumping from this frame

sliceSwitchON           = 0 # enable/disable slice level
                             switch

numSwitchPerFrame       = 0 # number of switches per frame
                            # when sliceSwitchON is enabled

numRestartMarkerPerSwitch = 0 # number of RST markers to
                              # decode per switch

ErrorConcealmentON       = 1 # Enable/Disable error
                             # concealment

debugTraceLevel          = 0 # Set debug trace level

lastNFramesToLog         = 0 # Number of frames to log debug
                             # trace if enabled
```

Note:

Please see Table 1-1 for the list of supported input and output chroma formats.

2.6 Standards Conformance and User-Defined Inputs

To check the conformance of the codec for the default input file shipped along with the codec, follow the steps as described in Section 2.4. To check the conformance of the codec for other input files of your choice, follow these steps:

- ❑ Copy the input files to the \Client\Test\TestVecs\Inputs sub-directory

- ☐ Copy the reference files to the \Client\Test\TestVecs\Reference subdirectory.

Edit the configuration file, TestVecs.cfg available in the \Client\Test\TestVecs\Config sub-directory. For details on the format of the TestVecs.cfg file, see Section 2.5.1.

2.7 Uninstalling the Component

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

Chapter 3

Sample Usage

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

3.1 Overview of the Test Application

The test application exercises the `IVIDDEC3` base class of the MJPEG Decoder library. The main test application files are jpegvdec_ti_hosttestapp.c and jpegvdec_rman_config.c. These files are available in the \IVAHD_001\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.

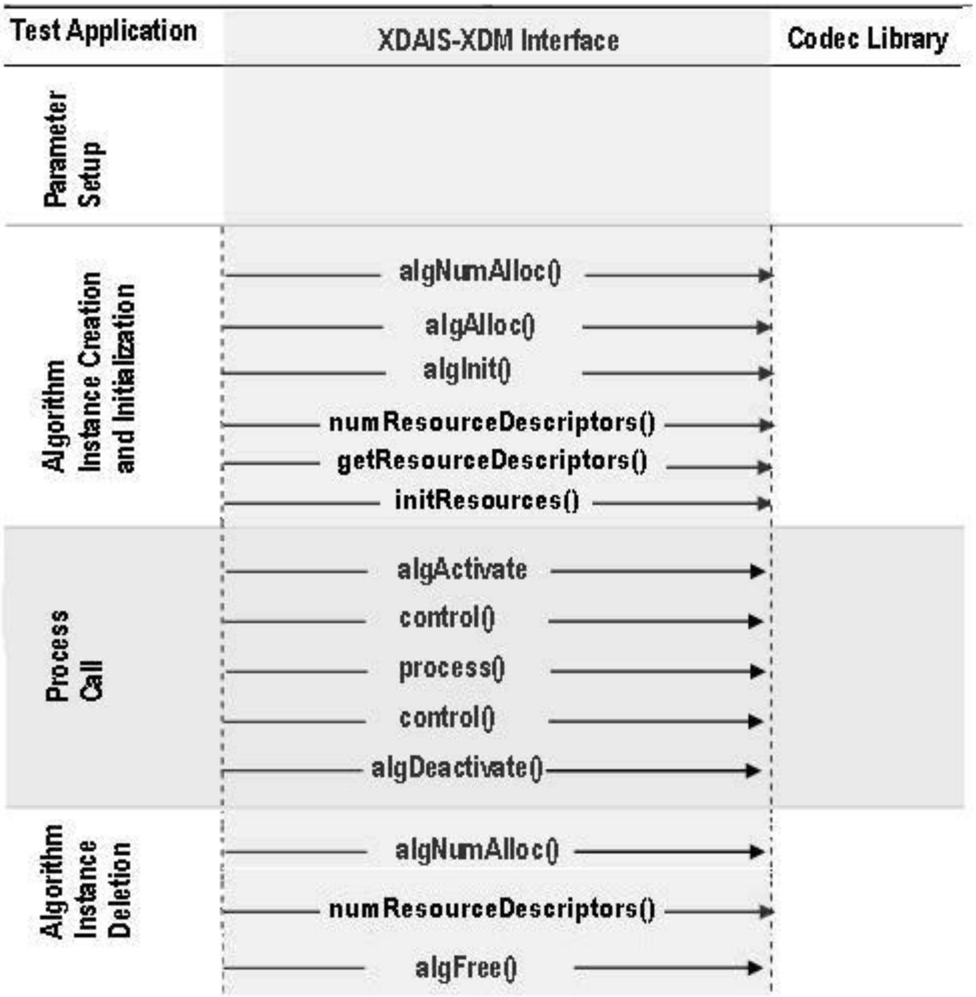


Figure 3-1 Test Application Sample Implementation

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:

1) 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.

2) 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 2.4.3.

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

4) 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:

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

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

3) `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:

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

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

6) `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:

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

2) 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.

3) 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 4.2.1.9). The inputs to the `process()`functions are input and output buffer descriptors, pointer to the `IVIDDEC3_InArgs` and `IVIDDEC3_OutArgs` structures.

4) On the call to the `process()` function for decoding a single frame of data, the software triggers the start of decode. After triggering the start of the decode frame, the video task can be put to `SEM-pend` state using semaphores. On receipt of interrupt signal at the end of frame 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.

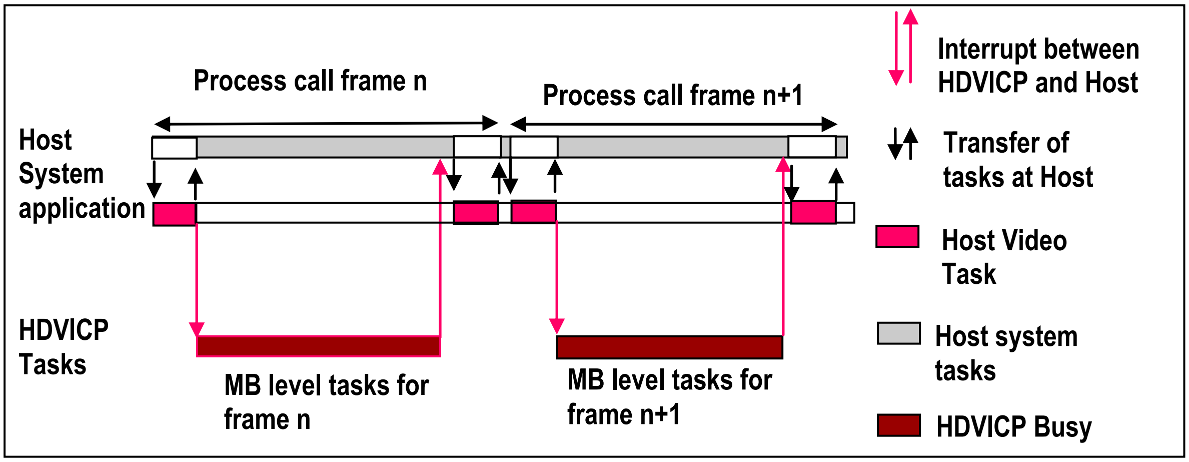


Figure 3-2. Process call with Host release

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:

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

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

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

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

9) `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:

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

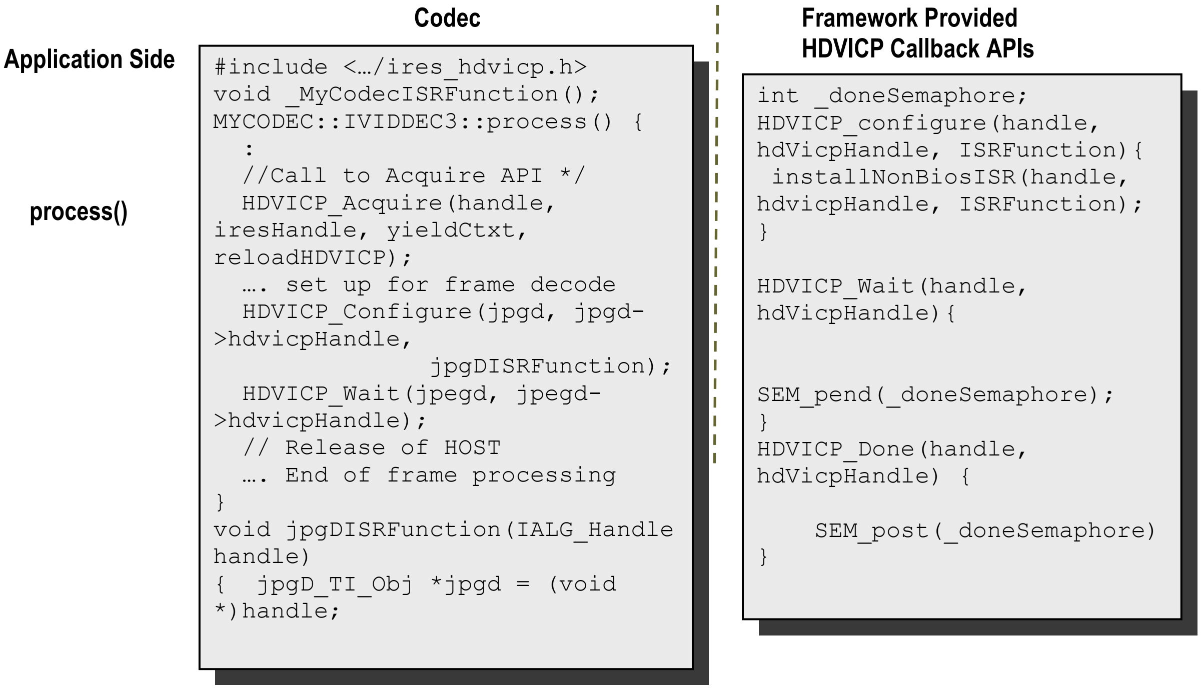
2) `algNumAlloc()` - To query the algorithm about the number of memory records it used.

3) `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 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.



```
#include <…/ires_hdvicp.h>
void _MyCodecISRFunction();
MYCODEC::IVIDDEC3::process() {
  :
  //Call to Acquire API */
  HDVICP_Acquire(handle,
iresHandle, yieldCtxt,
reloadHDVICP);
  …. set up for frame decode
  HDVICP_Configure(jpgd, jpgd-
>hdvicpHandle,
               jpgDISRFunction);
  HDVICP_Wait(jpegd, jpegd-
>hdvicpHandle);
  // Release of HOST
  …. End of frame processing
}
void jpgDISRFunction(IALG_Handle
handle)
{  jpgD_TI_Obj *jpgd = (void
*)handle;
```

```
int _doneSemaphore;
HDVICP_configure(handle,
hdVicpHandle, ISRFunction){
 installNonBiosISR(handle,
hdvicpHandle, ISRFunction);
}

HDVICP_Wait(handle,
hdVicpHandle){


SEM_pend(_doneSemaphore);
}
HDVICP_Done(handle,
hdVicpHandle) {

    SEM_post(_doneSemaphore)
}
```

Figure 3-3. 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.

3.3 Address Translations

The buffers addresses(DDR addresses) as seen by Ducati(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(HOSTCORTEXM3_OMAP4)
     #define VDMAVIEW_EXTMEM (0xFFFFFFFF)
#elif defined(HOSTCORTEXM3_NETRA)
     #define VDMAVIEW_EXTMEM (0xFFFFFFFF)
#else
     #define VDMAVIEW_EXTMEM (0x07FFFFFF)
#endif
```

3.4 Sample Test Application

The test application exercises the `IVIDDEC3` base class of the MJPEG Decoder.

Table 3-1 Process() Implementation

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

  TestApp_SetInitParams(&params.viddecParams);

  /*---------------- Decoder creation -----------------*/
  handle = (IALG_Handle) jpgVDEC_create();

  /* Optional: Set Run-time parameters in the Algorithm
via control() */
  jpgVDEC_control(handle, XDM_SETPARAMS);

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

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

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

     /* Get the status of the decoder using control */
     jpgVDEC_control(handle, XDM_GETSTATUS);

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


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

ALG_delete (handle);
```

Note:

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

This page is intentionally left blank

Chapter 4

API Reference

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

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 Datatypes

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
`IVIDEO_ContentType`	`IVIDEO_CONTENTTYPE_NA`	Content type is not applicable
	`IVIDEO_PROGRESSIVE` `IVIDEO_PROGRESSIVE_FRAME`	Progressive video content. Not applicable for MJPEG decoder.
	`IVIDEO_INTERLACED` `IVIDEO_INTERLACED_FRAME`	Interlaced video content. Not applicable for MJPEG decoder.
	`IVIDEO_INTERLACED_TOPFIELD`	Interlaced video content, Top field. Not applicable for MJPEG decoder.
	`IVIDEO_INTERLACED_BOTTOMFIELD`	Interlaced video content, Bottom field. Not applicable for MJPEG decoder.
	`IVIDEO_CONTENTTYPE_DEFAULT`	Default set to `IVIDEO_PROGRESSIVE`
`IVIDEO_FrameSkip`	`IVIDEO_NO_SKIP`	Do not skip the current frame. Not applicable for MJPEG decoder.
	`IVIDEO_SKIP_P`	Skip forward inter coded frame. Not applicable for MJPEG decoder.
	`IVIDEO_SKIP_B`	Skip bi-directional inter coded frame. Not applicable for MJPEG decoder.
	`IVIDEO_SKIP_I`	Skip intra coded frame. Not applicable for MJPEG decoder.
	`IVIDEO_SKIP_IP`	Skip I and P frame/field(s) Not applicable for MJPEG decoder.
	`IVIDEO_SKIP_IB`	Skip I and B frame/field(s). Not applicable for MJPEG decoder.
	`IVIDEO_SKIP_PB`	Skip P and B frame/field(s). Not applicable for MJPEG decoder.
	`IVIDEO_SKIP_IPB`	Skip I/P/B/BI frames Not applicable for MJPEG decoder.

Group or Enumeration Class	**Symbolic Constant Name**	**Description or Evaluation**
	`IVIDEO_SKIP_IDR`	Skip IDR Frame Not applicable for MJPEG decoder.
	`IVIDEO_SKIP_NONREFERENCE`	Skip non reference frame Not applicable for MJPEG decoder.
	`IVIDEO_SKIP_DEFAULT`	Default set to `IVIDEO_NO_SKIP`
`IVIDEO_VideoLayout`	`IVIDEO_FIELD_INTERLEAVED`	Buffer layout is interleaved. This enum is not applicable for MJPEG Decoder.
	`IVIDEO_FIELD_SEPARATED`	Buffer layout is field separated. This enum is not applicable for MJPEG Decoder.
	`IVIDEO_TOP_ONLY`	Buffer contains only top field. This enum is not applicable for MJPEG Decoder.
	`IVIDEO_BOTTOM_ONLY`	Buffer contains only bottom field. This enum is not applicable for MJPEG Decoder.
`IVIDEO_OperatingMode`	`IVIDEO_DECODE_ONLY`	Decoding Mode
	`IVIDEO_ENCODE_ONLY`	Encoding Mode. Not applicable for MJPEG decoder.
	`IVIDEO_TRANSCODE_FRAMELEVEL`	Transcode Mode of operation (encode/decode), which consumes /generates transcode information at the frame level. Not applicable for MJPEG decoder.
	`IVIDEO_TRANSCODE_MBLEVEL`	Transcode Mode of operation (encode/decode), which consumes /generates transcode information at the MB level. Not applicable for MJPEG decoder.
	`IVIDEO_TRANSRATE_FRAMELEVEL`	Transrate Mode of operation for encoder, which consumes transrate information at the frame level. Not applicable for MJPEG decoder.
	`IVIDEO_TRANSRATE_MBLEVEL`	Transrate Mode of operation for encoder, which consumes transrate information at the MB level. Not applicable for MJPEG decoder.
`IVIDEO_OutputFrameStatus`	`IVIDEO_FRAME_NOERROR`	Output buffer is available.
	`IVIDEO_FRAME_NOTAVAILABLE`	Codec does not have any output buffers.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`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. This enum is not applicable for MJPEG Decoder.
	`IVIDEO_I_PICTURE`	Intra coded picture. This enum is not applicable for MJPEG Decoder.
	`IVIDEO_P_PICTURE`	Forward inter coded picture. This enum is not applicable for MJPEG Decoder.
	`IVIDEO_B_PICTURE`	Bi-directional inter coded picture. This enum is not applicable for MJPEG Decoder.
`IVIDEO_DataMode`	`IVIDEO_FIXEDLENGTH`	Input to the decoder is in multiples of a fixed length (example, 4K) (input side for decoder).
	`IVIDEO_SLICEMODE`	Slice mode of operation (Input side for decoder).
	`IVIDEO_NUMROWS`	Number of MCU rows (output side for decoder).
	`IVIDEO_ENTIREFRAME`	Processing of entire frame data (default value)
`IVIDDEC3_displayDelay`	`IVIDDEC3_DISPLAY_DELAY_AUTO`	Decoder decides the display delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DECODE_ORDER`	Display frames are in decoded order without delay. Not applicable for MJPEG decoder.
	`IVIDDEC3_DISPLAY_DELAY_1`	Display the frames with 1 frame delay. Not applicable for MJPEG decoder.
	`IVIDDEC3_DISPLAY_DELAY_2`	Display the frames with 2 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_3`	Display the frames with 3 frame delay. Not supported in this version of MJPEG Decoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`IVIDDEC3_DISPLAY_DELAY_4`	Display the frames with 4 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_5`	Display the frames with 5 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_6`	Display the frames with 6 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_7`	Display the frames with 7 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_8`	Display the frames with 8 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_9`	Display the frames with 9 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_10`	Display the frames with 10 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_11`	Display the frames with 11 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_12`	Display the frames with 12 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_13`	Display the frames with 13 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_14`	Display the frames with 14 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_15`	Display the frames with 15 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAY_DELAY_16`	Display the frames with 16 frame delay. Not supported in this version of MJPEG Decoder.
	`IVIDDEC3_DISPLAYDELAY_DEFAULT`	Same as `IVIDDEC3_DISPLAY_DELAY_AUTO`. Not supported in this version of MJPEG Decoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
`XDM_DataFormat`	`XDM_BYTE`	Big endian stream (default value)
	`XDM_LE_16`	16-bit little endian stream. Not supported in this version of MJPEG Decoder.
	`XDM_LE_32`	32-bit little endian stream. Not supported in this version of MJPEG Decoder.
	`XDM_LE_64`	64-bit little endian stream. Not supported in this version of MJPEG Decoder.
	`XDM_BE_16`	16-bit big endian stream. Not supported in this version of MJPEG Decoder.
	`XDM_BE_32`	32-bit big endian stream. Not supported in this version of MJPEG Decoder.
	`XDM_BE_64`	64-bit big endian stream. Not supported in this version of MJPEG Decoder.
`XDM_ChromaFormat`	`XDM_YUV_420P`	YUV 4:2:0 planar. Supported for non-interleaved inputs in this version of MJPEG Decoder.
	`XDM_YUV_422P`	YUV 4:2:2 planar. Supported for non-interleaved inputs in this version of MJPEG Decoder.
	`XDM_YUV_422IBE`	YUV 4:2:2 interleaved (big endian).
	`XDM_YUV_422ILE`	YUV 4:2:2 interleaved (little endian). Not supported in this version of MJPEG Decoder.
	`XDM_YUV_444P`	YUV 4:4:4 planar.
	`XDM_YUV_411P`	YUV 4:1:1 planar. Not supported in this version of MJPEG Decoder.
	`XDM_GRAY`	Gray format. Supported only for input.
	`XDM_RGB`	RGB color format. Supported for thumbnail output.
	`XDM_YUV_420SP`	YUV 4:2:0 chroma semi-planar (default value)

Group or Enumeration Class	**Symbolic Constant Name**	**Description or Evaluation**
	`XDM_ARGB8888`	ARGB8888 color format. Not supported in this version of MJPEG Decoder.
	`XDM_RGB555`	RGB555 color format. Not supported in this version of MJPEG Decoder.
	`XDM_RGB565`	RGB565 color format. Not supported in this version of MJPEG Decoder.
	`XDM_YUV_444ILE`	YUV 4:4:4 interleaved (little endian) color format. Not supported in this version of MJPEG 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 MJPEG 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.
	`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

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`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 MJPEG 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 1 - applied concealment 0 – Error not found
	`XDM_INSUFFICIENTDATA`	Bit 10 1 - Insufficient data 0 – Error not found
	`XDM_CORRUPTEDDATA`	Bit 11 1 - Data problem/corruption 0 - Error not found
	`XDM_CORRUPTEDHEADER`	Bit 12 1 - Header problem/corruption 0 - Error not found
	`XDM_UNSUPPORTEDINPUT`	Bit 13 1 - Unsupported feature/parameter in input 0 - Error not found
	`XDM_UNSUPPORTEDPARAM`	Bit 14 1 - Unsupported input parameter or configuration 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`XDM_FATALERROR`	Bit 15 1 - Fatal error 0 - Recoverable error
`IJPEGDEC_ExtendedErrorCodes`	`IJPEGDEC_ERR_UNSUPPORTED_VIDDEC3PARAMS`	Bit 0 This error code has been deprecated.
	`IJPEGDEC_ERR_UNSUPPORTED_VIDDEC3DYNAMICPARAMS`	Bit 1 1 - Unsupported VIDDEC3DYNAMICPARAMS have been passed to the codec 0 - Error not found
	`IJPEGDEC_ERR_UNSUPPORTED_JPEGDECDYNAMICPARAMS`	Bit 2 1 - Unsupported JPEGDECDYNAMICPARAMS (i.e., extended) have been passed to the codec 0 - Error not found
	`IJPEGDEC_ERR_NOSLICE`	Bit 3 1 - Image does not have any slices and application is using slice level decoding. 0 - Error not found
	`IJPEGDEC_ERR_MBDATA`	Bit 4 1 - Invalid Input in MB data 0 - Error not found
	`IJPEGDEC_ERR_STANDBY`	Bit 5 1- HDVICP was not in standby when given to codec 0 - Error not found
	`IJPEGDEC_ERR_INVALID_MBOX_MESSAGE`	Bit 6 1 - Invalid MailBox Message has been received 0 - Error not found
	`IJPEGDEC_ERR_HDVICP_RESET`	Bit 7 1 – HDVICP is not put into RESET mode successfully 0 - Error not found
	`IJPEGDEC_ERR_HDVICP_WAIT_NOT_CLEAN_EXIT`	Bit 16 1 - Exit from HDVICP2 is not clean 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`IJPEGDEC_ERR_FRAME_HDR`	Bit 17 1 – Invalid Frame Header Information in the Input Stream which is passed to the codec. 0 - Error not found
	`IJPEGDEC_ERR_SCAN_HDR`	Bit 18 1 – Invalid Scan Header parameters in the Input Stream which is passed to the codec. 0 - Error not found
	`IJPEGDEC_ERR_HUFF_TBL_HDR`	Bit 19 1 – Invalid Huffman table Header parameters in the Input Stream which is passed to the codec. 0 - Error not found
	`IJPEGDEC_ERR_QUANT_TBL_HDR`	Bit 20 1 – Invalid Quantization table Header parameters in the Input Stream which is passed to the codec. 0 - Error not found
	`IJPEGDEC_ERR_OUTCHROMAFORMAT`	Bit 21 1 – Not supported output chroma format set by the application to the codec 0 - Error not found
	`IJPEGDEC_ERR_UNSUPPORTED_MARKER`	Bit 22 1 – Un Supported Marker in the Input stream. 0 - Error not found
	`IJPEGDEC_ERR_THUMBNAIL`	Bit 23 1 – Error in JFIF thumbnail marker. 0 - Error not found
	`IJPEGDEC_ERR_IRES_HANDLE`	Bit 24 1 – Handle provided the Resource Manager is NULL. 0 - Error not found
	`IJPEGDEC_ERR_DYNAMIC_PARAMS_HANDLE`	Bit 25 1 - Dynamic Params pointer passed to codec is NULL 0 - Error not found
	`IJPEGDEC_ERR_DATASYNC`	Bit 26 1 – Data Sync Error 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`IJPEGDEC_ERR_DOWNSAMPLE_INPUT_FORMAT`	Bit 27 1 – Scaling/Downsampling has been enabled for unsupported chroma format combination 0 - Error not found
	`IJPEGDEC_ERR_NOT_SUPPORTED_FEATURE`	Bit 28 1 – Scaling/Downsampling or Thumbnail mode of decoding has been enabled when slice level decoding is ON. 0 - Error not found
	`IJPEGDEC_ERR_NOT_SUPPORTED_RESOLUTION`	Bit 29 1 – Unsupported Width/Height are given to the codec. 0 - Error not found
`IjpegVDEC_ErrorStatus`	`JPEG_DECODE_THUMBNAIL_ERROR`	Bit 0 of `extendedErrorCode0` 1 – Unsupported value passed to codec for 'decodeThumbnail' parameter 0 - Error not found
	`JPEG_DYNAMIC_PARAMS_HANDLE_ERROR`	Bit 1 of `extendedErrorCode0` 1 - Dynamic Params pointer passed to codec is NULL 0 - Error not found
	`JPEG_THUMBNAIL_MODE_ERROR`	Bit 2 of `extendedErrorCode0` 1 - Unsupported value passed to codec for 'thumbnailMode' parameter 0 - Error not found
	`JPEG_DOWNSAMPLING_FACTOR_ERROR`	Bit 3 of `extendedErrorCode0` 1 - Unsupported value passed to codec for 'downsamplingFactor ' parameter 0 - Error not found
	`JPEG_STREAMING_COMPLIANT_ERROR`	Bit 4 of `extendedErrorCode0` 1 - Unsupported value passed to codec for 'streamingCompliant ' parameter 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`JPEG_NON_INTERLEAVED_STREAMING_COMPLIANT_ERROR`	Bit 5 of `extendedErrorCode0` 1 - 'streamingCompliant ' enabled for a non-interleaved image 0 - Error not found
	`JPEG_DECODE_HEADER_ERROR`	Bit 6 of `extendedErrorCode0` 1 - Unsupported value passed to codec for 'decodeHeader ' dynamic parameter 0 - Error not found
	`JPEG_DISPLAY_WIDTH_ERROR`	Bit 7 of `extendedErrorCode0` 1 - Unsupported value passed to codec for 'displayWidth ' dynamic parameter 0 - Error not found
	`JPEG_DYNAMIC_PARAMS_SIZE_ERROR`	Bit 8 of `extendedErrorCode0` 1 - Unsupported value passed to codec for 'size ' parameter of dynamic parmeters 0 - Error not found
	`JPEG_NULL_INSTANCE_HANDLE_ERROR`	Bit 9 of `extendedErrorCode0` 1 – Instance handle passed as NULL 0 - Error not found
	`JPEG_NULL_INARGS_POINTER_ERROR`	Bit 10 of `extendedErrorCode0` 1 – InArgs pointer passed as NULL in process call 0 - Error not found
	`JPEG_NULL_OUTARGS_POINTER_ERROR`	Bit 11 of `extendedErrorCode0` 1 – OutArgs pointer passed as NULL in process call 0 - Error not found
	`JPEG_NULL_INPUT_BUF_DESC_ERROR`	Bit 12 of `extendedErrorCode0` 1 – inbufdesc pointer passed as NULL in process call 0 - Error not found
	`JPEG_NULL_OUTPUT_BUF_DESC_ERROR`	Bit 13 of `extendedErrorCode0` 1 – outbufdesc pointer passed as NULL in process call 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`JPEG_INVALID_INARGS_SIZE`	Bit 14 of `extendedErrorCode0` 1 – Invalide 'size' parmeter for inArgs passed in process call 0 - Error not found
	`JPEG_INVALID_OUTARGS_SIZE`	Bit 15 1 – Invalide 'size' parameter for outArgs passed in process call 0 - Error not found
	`JPEG_NULL_INPUT_BUFFER_POINTER_ERROR`	Bit 16 of `extendedErrorCode0` 1 – Input buffer passed is NULL 0 - Error not found
	`JPEG_NULL_OUTPUT_BUF_DESC_POINTER_ERROR`	Bit 17 of `extendedErrorCode0` 1 – pointer to outArgs->displaybufs passed is NULL 0 - Error not found
	`JPEG_INVALID_NUM_OF_INPUT_BUFFERS_ERROR`	Bit 18 of `extendedErrorCode0` 1 – Invalid number of input buffers passed 0 - Error not found
	`JPEG_INVALID_INPUT_BYTES_ERROR`	Bit 19 of `extendedErrorCode0` 1 – Invalid input buffer size 0 - Error not found
	`JPEG_INVALID_INPUT_BUFFER_MEMORY_TYPE_ERROR`	Bit 20 of `extendedErrorCode0` 1 – Unsupported memory region type for input buffer 0 - Error not found
	`JPEG_INVALID_NUM_OF_OUTPUT_BUFFERS_ERROR`	Bit 21 of `extendedErrorCode0` 1 – Invalid number of output buffers 0 - Error not found
	`JPEG_NULL_OUTPUT_BUFFER_POINTER0_ERROR`	Bit 22 of `extendedErrorCode0` 1 – Output buffer -0 is passed as NULL to the codec 0 - Error not found
	`JPEG_INVALID_OUTPUT_BUFFER0_SIZE_ERROR`	Bit 23 of `extendedErrorCode0` 1 – Output buffer -0 size is invalid 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`JPEG_INVALID_OUTPUT_BUFFER0_MEMTYPE_ERROR`	Bit 24 of `extendedErrorCode0` 1 – Unsupported memory region passed for Output buffer -0 0 - Error not found
	`JPEG_NULL_OUTPUT_BUFFER_POINTER1_ERROR`	Bit 25 of `extendedErrorCode0` 1 – Output buffer -1 is passed as NULL to the codec 0 - Error not found
	`JPEG_INVALID_OUTPUT_BUFFER1_SIZE_ERROR`	Bit 26 of `extendedErrorCode0` 1 – Output buffer -1 size is invalid 0 - Error not found
	`JPEG_INVALID_OUTPUT_BUFFER1_MEMTYPE_ERROR`	Bit 27 of `extendedErrorCode0` 1 – Unsupported memory region passed for Output buffer -1 0 - Error not found
	`JPEG_NULL_OUTPUT_BUFFER_POINTER2_ERROR`	Bit 28 of `extendedErrorCode0` 1 – Output buffer -2 is passed as NULL to the codec 0 - Error not found
	`JPEG_INVALID_OUTPUT_BUFFER2_SIZE_ERROR`	Bit 29 of `extendedErrorCode0` 1 – Output buffer -2 size is invalid 0 - Error not found
	`JPEG_INVALID_OUTPUT_BUFFER2_MEMTYPE_ERROR`	Bit 30 of `extendedErrorCode0` 1 – Unsupported memory region passed for Output buffer -2 0 - Error not found
	`JPEG_INVALID_INPUT_ID_ERROR`	Bit 31 of `extendedErrorCode0` 1 – Invalid inputID passed to process call 0 - Error not found
	`JPEG_NUM_VDMA_DESC_EXCEEDS_ERROR`	Bit 0 of `extendedErrorCode1` 1 – Error in VDMA open 0 - Error not found
	`JPEG_INVALID_SOI_MARKER_ERROR`	Bit 1 of `extendedErrorCode1` 1 – No start of image (SOI) maker found in the input stream 0 - Error not found

Group or Enumeration Class	**Symbolic Constant Name**	**Description or Evaluation**
	`JPEG_INVALID_MARKER_SEG_LENGTH_ERROR`	Bit 2 of `extendedErrorCode1` 1 – Invalid marker segment length 0 - Error not found
	`JPEG_NON_STANDARD_MARKER_CODE_ERROR`	Bit 3 of `extendedErrorCode1` 1 – Marker Code is invalid 0 - Error not found
	`JPEG_INVALID_QUANT_TABLE_TYPE_ERROR`	Bit 4 of `extendedErrorCode1` 1 – Number of Q tables in DQT is more than supported 0 - Error not found
	`JPEG_QUANT_TABLE_BYTES_READ_ERROR`	Bit 5 of `extendedErrorCode1` 1 – Error in Q table reading 0 - Error not found
	`JPEG_INVALID_HUFFMAN_TABLE_TYPE_ERROR`	Bit 6 of `extendedErrorCode1` 1 – Error in Huffman table reading 0 - Error not found
	`JPEG_HUFFMAN_CODE_LENGTH_SIZE_EXCEED_ERROR`	Bit 7 of `extendedErrorCode1` 1 – Error in Huffman table code length 0 - Error not found
	`JPEG_HUFFMAN_TABLE_MARKER_SEG_SIZE_ERROR`	Bit 8 of `extendedErrorCode1` 1 – Error in Huffman table marker syntax 0 - Error not found
	`JPEG_HUFFMAN_TABLE_BYTES_READ_ERROR`	Bit 9 of `extendedErrorCode1` 1 – Error in Huffman table number of bytes to be read 0 - Error not found
	`JPEG_INVALID_SAMPLE_PRECISION_ERROR`	Bit 10 of `extendedErrorCode1` 1 – Error in sample precision (only 8-bit samples are supported) 0 - Error not found
	`JPEG_INVALID_NUM_COMPONENTS_ERROR`	Bit 11 of `extendedErrorCode1` 1 – Unsupported number of components in the header 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`JPEG_FRAME_HDR_BYTES_READ_ERROR`	Bit 12 of `extendedErrorCode1` 1 – Error in frame header bytes 0 - Error not found
	`JPEG_NOT_SUPPORTED_FORMAT_ERROR`	Bit 13 of `extendedErrorCode1` 1 – Unsupported chroma format 0 - Error not found
	`JPEG_ARITHMETIC_DECODING_NOT_SUPPORTED_MARKER_ERROR`	Bit 14 of `extendedErrorCode1` 1 – Arithmetic decoding found, which is not supported 0 - Error not found
	`JPEG_PROG_DECODING_NOT_SUPPORTED_MARKER_ERROR`	Bit 15 of `extendedErrorCode1` 1 – Arithmetic ext decoding found, which is not supported 0 - Error not found
	`JPEG_LOSSLESS_DECODING_NOT_SUPPORTED_MARKER_ERROR`	Bit 16 of `extendedErrorCode1` 1 –Lossless decoding found, which is not supported 0 - Error not found
	`JPEG_DIFFERENTIAL_DECODING_NOT_SUPPORTED_MARKER_ERROR`	Bit 17 of `extendedErrorCode1` 1 –Differential decoding found, which is not supported 0 - Error not found
	`JPEG_JFIF_THUMBNAIL_IDENTIFIER_ERROR`	Bit 18 of `extendedErrorCode1` 1 –Error in JFIF identifier 0 - Error not found
	`JPEG_JFIF_THUMBNAIL_BYTES_READ_ERROR`	Bit 19 of `extendedErrorCode1` 1 –Error in JFIF bytes 0 - Error not found
	`JPEG_JFIF_EXTN_NO_SOI_ERROR`	Bit 20 of `extendedErrorCode1` 1 –SOI not found in JFIF extension 0 - Error not found
	`JPEG_JFIF_NOT_SUPPORTED_FEATURE_ERROR`	Bit 21 of `extendedErrorCode1` 1 –Unsupported JFIF extension found 0 - Error not found

Group or Enumeration Class	**Symbolic Constant Name**	**Description or Evaluation**
	`JPEG_FORCECHROMA_OUTPUTCHROMA_FORMAT_MISMATCH_ERROR`	Bit 22 of `extendedErrorCode1` 1 –Unsupported force chroma format selected for the given input image 0 - Error not found
	`JPEG_INVALID_VERT_SCAN_FREQ_ERROR`	Bit 23 of `extendedErrorCode1` 1 –Error in vertical scan frequency for one of the components 0 - Error not found
	`JPEG_INVALID_HORI_SCAN_FREQ_ERROR`	Bit 24 of `extendedErrorCode1` 1 –Error in horizontal scan frequency for one of the components 0 - Error not found
	`JPEG_INVALID_QUANT_DEST_SELECTOR_ERROR`	Bit 25 of `extendedErrorCode1` 1 –Error in Q table ID for one of the components 0 - Error not found
	`JPEG_DC_ENTROPY_CODING_DEST_ERROR`	Bit 26 of `extendedErrorCode1` 1 –Error in scan header parsing- DC component 0 - Error not found
	`JPEG_AC_ENTROPY_CODING_DEST_ERROR`	Bit 27 of `extendedErrorCode1` 1 –Error in scan header parsing- AC component 0 - Error not found
	`JPEG_ECD_VLD_OUT_OF_TABLE_ERROR`	Bit 28 of `extendedErrorCode1` 1 – ECD error: vld out of table 0 - Error not found
	`JPEG_ECD_RESTART_INTERVAL_ERROR`	Bit 29 of `extendedErrorCode1` 1 – ECD error: invalid RST interval 0 - Error not found
	`JPEG_ECD_BLOCK_COEFF_NUM_ERROR`	Bit 30 of `extendedErrorCode1` 1 – ECD error: invalid number of coefficients 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`JPEG_GET_DATA_SYNC_NULL_FUNC_POINTER_ERROR`	Bit 31 of `extendedErrorCode1` 1 – parameter ‘getDataFxn’ in dynamic params is NULL 0 - Error not found
	`JPEG_PUT_DATA_SYNC_NULL_FUNC_POINTER_ERROR`	Bit 0 of `extendedErrorCode2` 1 – parameter ‘putDataFxn ‘ in dynamic params is NULL 0 - Error not found
	`JPEG_HDVICP_ACQUIRE_AND_CONFIGURE_ERROR`	Bit 1 of `extendedErrorCode2` 1 – Error in HDVICP acquire 0 - Error not found
	`JPEG_NULL_ALGORITHM_HANDLE_ERROR`	Bit 2 of `extendedErrorCode2` 1 – Algorithm handle provided is NULL 0 - Error not found
	`JPEG_GETVERSION_NULL_BUF_POINTER_ERROR`	Bit 3 of `extendedErrorCode2` 1 – Error in the buffer provided in GETVERSION through status->data 0 - Error not found
	`JPEG_IRES_RESOURCE_DESC_ERROR`	Bit 4 of `extendedErrorCode2` 1 – resource descriptor pointer passed through IRES interface is NULL 0 - Error not found
	`JPEG_IRES_RESOURCE_DESC_HANDLE_ERROR`	Bit 5 of `extendedErrorCode2` 1 – handle to a resource passed through IRES interface is NULL 0 - Error not found
	`JPEG_NULL_STATUS_DATA_BUF`	Bit 6 of `extendedErrorCode2` 1 – NULL buffer passed through status->data.buf field for GETVERSION call 0 - Error not found
	`JPEG_EXCEED_BYTES_CONSUMED_ERROR`	Bit 7 of `extendedErrorCode2` 1 – number of bytes consumed is more than total input bytes provided 0 - Error not found

Group or Enumeration Class	**Symbolic Constant Name**	**Description or Evaluation**
	`JPEG_INPUT_DATASYNC_NUMBLOCKS_ERROR`	Bit 8 of `extendedErrorCode2` 1 – unsupported number of blocks in input data sync 0 - Error not found
	`JPEG_INPUT_DATASYNC_BUFF_POINTER_ERROR`	Bit 9 of `extendedErrorCode2` 1 – base address for input data sync provided is NULL 0 - Error not found
	`JPEG_INPUT_DATASYNC_BLOCKSIZE_ERROR`	Bit 10 of `extendedErrorCode2` 1 – block size provided through input data sync is zero 0 - Error not found
	`JPEG_INPUT_DATASYNC_NOT_VALID`	Bit 11 of `extendedErrorCode2` 1 – unsupported combination of input data sync mode 0 - Error not found
	`JPEG_OUTPUT_DATASYNC_NUMBLOCKS_ERROR`	Bit 12 of `extendedErrorCode2` 1 – unsupported number of blocks for output data sync call 0 - Error not found
	`JPEG_SLICE_LEVEL_INPUT_NO_RST_MARKER_ERROR`	Bit 13 of `extendedErrorCode2` 1 – No RST marker found for slice level input data sync 0 - Error not found
	`JPEG_DOWNSAMPLING_IN_NON_TILED_ERROR`	Bit 14 of `extendedErrorCode2` 1 – Scaling/Downsampling has been enabled when the output buffer provided to codec is not in TILED region 0 - Error not found
	`JPEG_DOWNSAMPLING_NOT_SUPPORTED_FORMAT_ERROR`	Bit 15 of `extendedErrorCode2` 1 – Scaling/Downsampling has been enabled for unsupported chroma format combination 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`JPEG_DOWNSAMPLING_NOT_SUPPORTED_FEATURE_ERROR`	Bit 16 of `extendedErrorCode2` 1 – Scaling/Downsampling has been enabled when data sync or slice level decoding is enabled. 0 - Error not found
	`JPEG_THUMBNAIL_NOT_SUPPORTED_FEATURE_ERROR`	Bit 17 of `extendedErrorCode2` 1 – Thumbnail decoding has been enabled when when data sync or slice level decoding is enabled. 0 - Error not found
	`JPEG_NOT_SUPPORTED_WIDTH_ERROR`	Bit 18 of `extendedErrorCode2` 1 – unsupported MaximumWidth/MinimumWidth of Image is given to the codec . 0 - Error not found
	`JPEG_NOT_SUPPORTED_HEIGHT_ERROR`	Bit 19 of `extendedErrorCode2` 1 – unsupported MaximumHeight/MinimumHeigh of Image is given to the codec . 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. Not supported in this version of MJPEG 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`
- `XDM_DataSyncDesc`
- `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. Required only for tiled memory.
`height`	`XDAS_Int32`	Input	Height of buffer in 8-bit bytes. Required only for tiled memory.
`bytes`	`XDAS_Int32`	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 MJPEG Decoder, the buffer details are:

❑ Number of input buffers required is 1.

- ❑ Number of output buffers required is based on output chroma format.
- ❑ There is no restriction on input buffer size except that it should contain atleast one frame of encoded data.
- ❑ 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 XDM_DataSyncDesc

|| Description

This structure describes the chunk of data being transferred in one call to putData or getData.

|| Fields

Field	**Data Type**	**Input/ Output**	**Description**
`scatteredBlocksFlag`	`XDAS_Int32`	Input	Flag indicating whether the individual data blocks may be scattered in memory.
`baseAddr`	`XDAS_Int32 *`	Input	Base address of single data block or pointer to an array of data block addresses of size numBlocks.
`numBlocks`	`XDAS_Int32`	Input	Number of blocks available.
`varBlockSizesFlag`	`XDAS_Int32`	Input	Flag indicating whether any of the data blocks vary in size.
`blockSizes`	`XDAS_Int32 *`	Input	Variable block sizes array.

Note:

- ❑ The following parameters are not supported/updated (don’t care) in data sync at output side
- `scatteredBlocksFlag`
- `baseAddr`
- `varBlockSizesFlag`
- `blockSizes`
- ❑ There are three modes of operations in Data Sync at Input side
    - Slice Mode (`IVIDEO_SLICEMODE`)
    - Fixed Length Mode (`IVIDEO_FIXEDLENGTH`)
    - Entire Frame Mode (`IVIDEO_ENTIREFRAME`) (without Data Sync)
- ❑ In Slice Mode, the following conditions should be met.
- The input stream should contain RST marker
- `scatteredBlockFlag` should be TRUE
- `varBlockSizesFlag` may be TRUE/FALSE
- `numBlocks` can be any positive number between 1 to 32.
- Total size per Data Sync call should be >= page size (8192 bytes). If it is less than page size (8192 bytes), then it is assumed as the last data sync.
- ❑ In Fixed Length Mode, the following conditions should be met.
- `scatteredBlockFlag` should be FALSE
- `varBlockSizesFlag` should be FALSE
- `numBlocks` should be 1.
- During the first data sync call, the data provided need not to be multiple of page size (8192 bytes).
- Total size per data sync call (except the first call) should be multiple of page size (8192 bytes). If it is less than page size (8192 bytes), then it is assumed as the last data sync.
- ❑ There are two modes of operations in Data Sync at Output side
    - – NUMROWS Mode (`IVIDEO_NUMROWS`)
    - – Entire Frame Mode (`IVIDEO_ENTIREFRAME`) (without Data Sync)
- `numBlocks` is set by the codec. User need not set this parameter.

4.2.1.6 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]`	`XDM2_SingleBufDesc`	Input/ Output	Description for video planes
`metadataPlaneDesc [IVIDEO_MAX_NUM_METADATA_PLANES]`	`XDM2_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/ Output	Decoded image region including 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`.
`topFieldFirstFlag`	`XDAS_Int32`	Input/ Output	Indicates when the application (should display)/(had captured) the top field first. Not applicable for MJPEG decoder.
`repeatFirstFieldFlag`	`XDAS_Int32`	Input/ Output	Indicates when the first field should be repeated. Not applicable for MJPEG decoder
`frameStatus`	`XDAS_Int32`	Input/ Output	Video in/out buffer status.
`repeatFrame`	`XDAS_Int32`	Input/ Output	Number of times to repeat the displayed frame. Not applicable for MJPEG decoder.

Field	Data Type	Input/ Output	Description
`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. Not applicable for MJPEG decoder.
`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	Not applicable for MJPEG decoder
`rangeMappingChroma`	`XDAS_Int32`	Input/ Output	Not applicable for MJPEG decoder
`enableRangeReductionFlag`	`XDAS_Int32`	Input/ Output	ON/OFF, default is OFF. Not applicable for MJPEG decoder

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
- `repeatFirstFieldFlag`
- `repeatFrame`
- `scalingWidth`
- `scalingHeight`
- `rangeMappingLuma`
- `rangeMappingChroma`
- `enableRangeReductionFlag`

4.2.1.7 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. 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

4.2.1.8 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.
`maxHeight`	`XDAS_Int32`	Input	Maximum video height to be supported in pixels. The supported range is [32, 4096]. Default is 1088.
`maxWidth`	`XDAS_Int32`	Input	Maximum video width to be supported in pixels. The supported range is [32, 4096]. Default is 1920.
`maxFrameRate`	`XDAS_Int32`	Input	Maximum frame rate in fps * 1000 to be supported. Not applicable for MJPEG decoder.
`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. Not applicable for MJPEG decoder.

Field	**Data Type**	**Input/ Output**	**Description**
`dataEndianness`	`XDAS_Int32`	Input	Endianness of input data. See `XDM_DataFormat` enumeration for details. Default is `XDM_BYTE`.
`forceChromaFormat`	`XDAS_Int32`	Input	Sets the output to the specified format. See Table 1-1 for details. See `XDM_ChromaFormat` and `eChromaFormat_t` enumerations for details. Default value is `XDM_YUV_420SP`.
`operatingMode`	`XDAS_Int32`	Input	Video coding mode of operation (encode/decode/transcode/transrate). Only decode mode is supported in this version.
`displayDelay`	`XDAS_Int32`	Input	Display delay to start display. Not applicable for MJPEG decoder.
`inputDataMode`	`XDAS_Int32`	Input	Input mode of operation. For decoder, the supported values are `IVIDEO_FIXEDLENGTH`, `IVIDEO_SLICEMODE` and `IVIDEO_ENTIREFRAME`. Default value is `IVIDEO_ENTIREFRAME`.
`outputDataMode`	`XDAS_Int32`	Input	Output mode of operation. For decoder, the supported values are `IVIDEO_NUMROWS` and `IVIDEO_ENTIREFRAME`. Default value is `IVIDEO_ENTIREFRAME`.
`numInputDataUnits`	`XDAS_Int32`	Input	Number of input slices/buffers. This parameter is ignored by the decoder. Refer Chapter 8 for more details.
`numOutputDataUnits`	`XDAS_Int32`	Input	Number of output rows. For `IVIDEO_ENTIREFRAME` mode, it should set to 1.
`errorInfoMode`	`XDAS_Int32`	Input	Enable/disable packet error information for input/output. Not supported in this version of MJPEG decoder.
`displayBufsMode`	`XDAS_Int32`	Input	Indicates the `displayBufs` mode. This field can be set either as `IVIDDEC3_DISPLAYBUFS_EMBEDDED` or `IVIDDEC3_DISPLAYBUFS_PTRS`. Default value is `IVIDDEC3_DISPLAYBUFS_EMBEDDED`.
`metadataType`	`XDAS_Int32[]`	Input	Type of each metadata plane. Not supported.

Note:

- ❑ Maximum video height and width supported are 4096 pixels and 4096 pixels respectively.
- ❑ The minimum height and width supported is 32 pixels.
- ❑ `dataEndianness` field should be set to `XDM_BYTE`.

4.2.1.9 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**
`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: 0 (`XDM_DECODE_AU`) - Decode entire frame including all the headers 1 (`XDM_PARSE_HEADER`) - Decode only one NAL unit Default value is `XDM_DECODE_AU`.
`displayWidth`	`XDAS_Int32`	Input	If the field is set to: 0 - Uses decoded image width as pitch If any other value greater than the decoded image width is given, then this value in pixels is used as pitch. Default value is 0.
`frameSkipMode`	`XDAS_Int32`	Input	Frame skip mode. See `IVIDEO_FrameSkip` enumeration for details. Not applicable to MJPEG decoder.
`newFrameFlag`	`XDAS_Int32`	Input	Flag to indicate that the algorithm should start a new frame. Valid values are `XDAS_TRUE` and `XDAS_FALSE`. 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 MJPEG decoder.
`*putDataFxn`	`XDM_DataSyncPutFxn`	Input	`DataSync` call back function pointer for `putData`. Default value is `NULL`.
`putDataHandle`	`XDM_DataSyncHandle`	Input	`DataSync` handle for `putData`. Default value is `NULL`.

Field	Data Type	Input/ Output	Description
`*getDataFxn`	`XDM_DataSyncGetFxn`	Input	`DataSync` call back function pointer for `getData.` Default value is `NULL.`
`getDataHandle`	`XDM_DataSyncHandle`	Input	`DataSync` handle for `getData.` Default value is `NULL.`
`putBufferFxn`	`XDM_DataSyncPutBufferFxn`	Input	Not supported in this decoder.
`putBufferHandle`	`XDM_DataSyncHandle`	Input	Not supported in this decoder.
`lateAcquireArg`	`XDAS_Int32`	Input	Argument used during late acquire. Default value is `IRES_HDVICP2_UNKNOWNLATEACQUIREARG`.

Note:

- ❑ The `displayWidth` should be >= image width
- ❑ `displayWidth` should be 128 byte aligned for non-TILED output buffers.
- ❑ If the `displayWidth` is set to 0, the decoder uses the image width as `displayWidth.`
- ❑ The default value of `displayWidth` is 0.

4.2.1.10 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. MJPEG Decoder simply copies the inputID value to the outputID value of IVIDDEC3_OutArgs structure.

4.2.1.11 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. Not Supported in this version of MJPEG decoder.
`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’. Not applicable to MJPEG decoder.
`bitRate`	`XDAS_Int32`	Output	Average bit-rate in bits per second. Not applicable to MJPEG decoder.

Field	Data Type	Input/ Output	Description
`contentType`	`XDAS_Int32`	Output	Video content. See `IVIDEO_ContentType` enumeration for details.
`sampleAspectRatioHeight`	`XDAS_Int32`	Output	Sample aspect ratio for height. Not supported.
`sampleAspectRatioWidth`	`XDAS_Int32`	Output	Sample aspect ratio for width. Not supported.
`bitRange`	`XDAS_Int32`	Output	Bit range. It is set to `IVIDEO_YUVRANGE_FULL`.
`forceChromaFormat`	`XDAS_Int32`	Output	Output chroma format. See `XDM_ChromaFormat` enumeration for details.
`operatingMode`	`XDAS_Int32`	Output	Mode of operation: Encoder/Decoder/Transcode/Transrate. This decoder supports `IVIDEO_DECODE_ONLY` only.
`frameOrder`	`XDAS_Int32`	Output	Indicates the output frame order. See `IVIDDEC3_displayDelay` enumeration for more details. Not applicable to MJPEG decoder.
`inputDataMode`	`XDAS_Int32`	Output	Input mode of operation. For decoder, it is fixed length/slice mode/entire frame.
`outputDataMode`	`XDAS_Int32`	Output	Output mode of operation. For decoder, it is the row mode/entire frame.
`bufInfo`	`XDM_AlgBufInfo`	Output	Input and output buffer information. See `XDM_AlgBufInfo` data structure for details.
`numInputDataUnits`	`XDAS_Int32`	Output	Decoder will set to appropriate value from the `IVIDDEC3_Params` structure mentioned above.
`numOutputDataUnits`	`XDAS_Int32`	Output	Decoder will set to appropriate value from the `IVIDDEC3_Params` structure mentioned above.
`configurationID`	`XDAS_Int32`	Output	Decoder will set it to 1.
`metadataType`	`XDAS_Int32[]`	Input	Type of each metadata plane. Not supported in this decoder.

Field	Data Type	Input/ Output	Description
`decDynamicParams`	`IVIDDEC3_DynamicParams`	Output	Current values of the decoder's dynamic parameters.

4.2.1.12 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` A value of zero (0) indicates an invalid ID. The first zero entry in array will indicate end of valid `outputID`s 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. 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.

Field	**Data Type**	**Input/ Output**	**Description**
			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 `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 MJPEG Decoder Data Structures

This section includes the following MJPEG Decoder specific data structures:

- `IJPEGVDEC_Params`
- `IJPEGVDEC_DynamicParams`
- `IJPEGVDEC_InArgs`
- `IJPEGVDEC_Status`
- `IJPEGVDEC_OutArgs`

4.2.2.1 IJPEGVDEC_Params

|| Description

This structure defines the creation parameters and any other implementation specific parameters for an MJPEG 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 (`IJPEGVDEC_EC_ENABLE`) to enable error concealment And 0 (`IJPEGVDEC_EC_DISABLE)` to disable error concealment. Default value is `IJPEGVDEC_EC_DISABLE.` Error concealment is supported for YUV420 interleaved inputs only. Please note that decoding takes more cycles with error concealment enabled than normal decoding.
`debugTraceLevel`	`XDAS_UInt32`	Input	Specifies the debug trace level. MJPEG Decoder supports till level 4. Each higher level logs more debug trace data. Default value is 0.

Field	**Data Type**	**Input/ Output**	**Description**
`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. Valid only if `debugTraceLevel` is greater than 0. Default value is 0.
`sliceSwitchON`	`XDAS_UInt32`	Input	Set it to 1 to enable slice level decoding feature and 0 to enable entire frame decoding feature. Default value is 0.
`numSwitchPerFrame`	`XDAS_UInt32`	Input	Specifies the number of times process call will be called to decode a frame. Valid only if “sliceSwitchON” is 1. Default value is 0.
`numRestartMarkerPerSwitch`	`XDAS_UInt32`	Input	Specifies the number of slices to be decoded per switch. Valid only if “sliceSwitchON” is 1. Default value is 0.

Note:

For handling slice level decoding , three extended create time parameters “sliceSwitchON”, “numSwitchPerFrame” and “numRestartMarkerPerSwitch” are added in the Params struct. If the sliceSwitchON parameter is set to 1, the decoder decodes in slice mode depending on the mode which is being selected. If set to 0, the decoder decodes the full image.

When the sliceSwitchON parameter is set to 1, the numSwitchPerFrame parameter specifies the number of times process call will be called to decode a frame.

When the sliceSwitchON parameter is set to 1, the numRestartMarkerPerSwitch parameter specifies the number of slices to be decoded per switch.

When both “numRestartMarkerPerSwitch” and “numSwitchPerFrame” are having non-zero value, “numSwitchPerFrame” will be considered as high priority and “numRestartMarkerPerSwitch” will be discarded.

For a more detailed description of the slice level decoding feature, refer to chapter-10 of this user guide.

4.2.2.2 IJPEGVDEC_DynamicParams

|| Description

This structure defines the run-time parameters and any other implementation specific parameters for an MJPEG Decoder 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.
`decodeThumbnail`	`XDAS_Int32`	Input	If set to 1, decodes thumbnail image and dumps the output in display buffer. If set to ‘0’, decodes the original image (not thumbnail image) and dumps the output in display buffer. Thumbnail decoding is not supported when data sync or slice level decoding is enabled. Default value is 0.
`thumbnailMode`	`XDAS_Int32`	Input	Supported thumbnail modes are `THUMBNAIL_JFIF` (decode and output thumbnail present in JFIF marker), `THUMBNAIL_EXIF` (decode and output thumbnail present in Exif marker) and `IJPEGVDEC_THUMBNAIL_DOWNSAMPLE`. Default value is `IJPEGVDEC_THUMBNAIL_DOWNSAMPLE.`

Field	**Data Type**	**Input/ Output**	**Description**
`downsamplingFactor`	`XDAS_Int32`	Input	Scaling factor. Supported values are `IJPEGVDEC_NODOWNSAMPLE`, `IJPEGVDEC_DOWNSAMPLEBY2` and `IJPEGVDEC_DOWNSAMPLEBY4`. Downsampling is supported only for YUV444 and GRAY input chroma formats and only in TILER Mode. In addition, the output chroma format must be set to YUV444 Planar for YUV444 input and YUV420 semi-planar for Grayscale input. In addition, downsampling is not supported when data sync or slice level decoding is enabled. Default value is `IJPEGVDEC_NODOWNSAMPLE`.
`streamingCompliant`	`XDAS_Int32`	Input	If an Input Image is Non-Interleaved, the application has to set this params to "0" (DISABLE), if it is Interleaved, value will be "1" (ENABLE). This Parameter along with forceChromaFormat determines whether we have to give Planar Buffers from GETBUFINFO control call. Default value is ENABLE.

Note:

For handling thumbnails, three extended parameters “decodeThumbnail”, “thumbnailMode” and “downsamplingFactor” are added in the DynamicParams struct. If the decodeThumbnail parameter is set to 1, the decoder decodes only the thumbnail. If set to 0, the decoder decodes the full image (if thumbnail is present in the encoded stream, the decoder skips it).

When the decodeThumbnail parameter is set to 1, the thumbnailMode parameter specifies the type of thumbnail: `IJPEGVDEC_THUMBNAIL_JFIF`, `IJPEGVDEC_THUMBNAIL_EXIF` or `IJPEGVDEC_THUMBNAIL_DOWNSAMPLE`. In JFIF, the thumbnail could be RGB or JPEG. The decoder does not support RGB palette (1 byte per pixel) thumbnails. Thumbnail decoding is not supported when data sync or slice level decoding is enabled.

Downsampling is supported only for YUV444 and GRAY chroma formats and only in TILER Mode. In addition, the output chroma format must be set to YUV444 Planar for YUV444 input and YUV420 semi-planar for Grayscale input. In addition, downsampling is not supported when data sync or slice level decoding is enabled.

4.2.2.3 IJPEGVDEC_InArgs

‖ Description

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

‖ Fields

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

4.2.2.4 IJPEGVDEC_Status

‖ Description

This structure defines parameters that describe the status of the MJPEG 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. Refer to *IjpegVDEC_ErrorStatus*
`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
`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. MJPEG Decoder supports till level 4. Each higher level logs more debug trace data.

Field	Data Type	Input/ Output	Description
`lastNFramesToLog`	`XDAS_UInt32`	Output	Specifies the number of most recent frames logged 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 IJPEGVDEC_OutArgs

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
`viddec3OutArgs`	`IVIDDEC3_OutArgs`	Output	See `IVIDDEC3_OutArgs` data structure for details.
`IsGrayFlag`	`XDAS_UInt32`	Output	This is set if the input to the decoder is a grayscale image. For 420 and Gray scale images, the output chroma format is 420SP. This flag will differentiate the MCU size required in output data sync usage. If IsGrayFlag is set to 1, the row size is 8xWidth otherwise rowsize is 16xWidth.

4.3 Interface Functions

This section describes the application programming interfaces used in the MJPEG Decoder. The MJPEG 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 MJPEG 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 MJPEG 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 MJPEG 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 MJPEG decoder
instance */

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

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

IVIDDEC3_Status *status /* MJPEG 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 MJPEG 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 MJPEG 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 MJPEG 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 MJPEG 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 MJPEG decoder
runtime input arguments data structure */
```

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

|| Return Value

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

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

|| Description

This function does the basic MJPEG video decoding. The first argument to `process()` is a handle to the MJPEG 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 MJPEG 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 MJPEG 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 MJPEG 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()`

This page is intentionally left blank

Frequenty Asked Questions

This section answers frequently asked questions related to using MJPEG 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.
Make file build fails	Make sure you have set environment variable <CG_TOOL_DIR> as defined in section2.3. Make sure gmake utility path is added to PATH environment variable as mentioned in section 2.3

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). 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 MJPEG 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.

Question	**Answer**
Does this MJPEG Decoder support custom quantization tables?	Yes.
Does this MJPEG Decoder support custom Huffman tables?	Yes.
Does Algorithm support DataSync mechanism for low-delay applications?	Yes. It has the mechanism for both input and output buffers.
Does this decoder support “decode header only” feature?	Yes.
Does this decoder support decoding of thumbnails?	Yes. The decoder supports decoding of thumbnails present in JFIF and Exif markers.
How does the decoder handle APPx markers other than JFIF and Exif?	The decoder just skips APPx markers other than JFIF and Exif.
What are the maximum and minimum resolutions supported by the decoder?	This decoder supports resolutions ranging from 32x32 to 4096x4096.
What are the chroma formats supported for input and output?	Please see Table 1-1.
Does the decoder support decoding of multiple scan JPEGs?	Yes.
Does the decoder support slice level decoding?	Yes.
Does the decoder support thumbnail decoding at slice level?	No.
Does the decoder support decoding of multiple scan JPEGs in slice level decoding?	No.
Does the decoder support Data Sync mechanism in slice level decoding?	No. Support for data sync and slice level decoding are mutually exclusive.
Does the decoder support error concealment?	Yes.
What are the input chroma formats for which error concealment is supported?	Error concealment is supported for YUV420 interleaved (i.e., single scan) inputs only.

Question	**Answer**
Will the decoding time depends on error concealment enabled/disabled?	Yes. Decoding takes more time with error concealment enabled. Significant performance impact will be observed with EC enabled.

This page is intentionally left blank

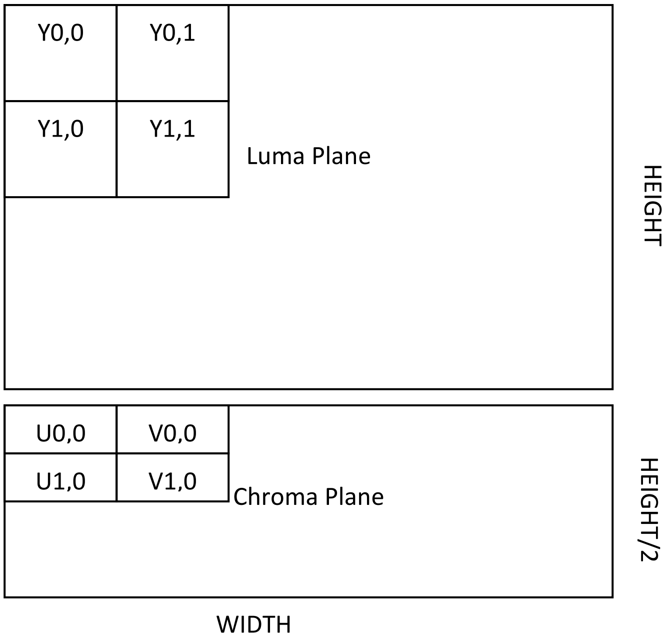
Chapter 6

Picture Format

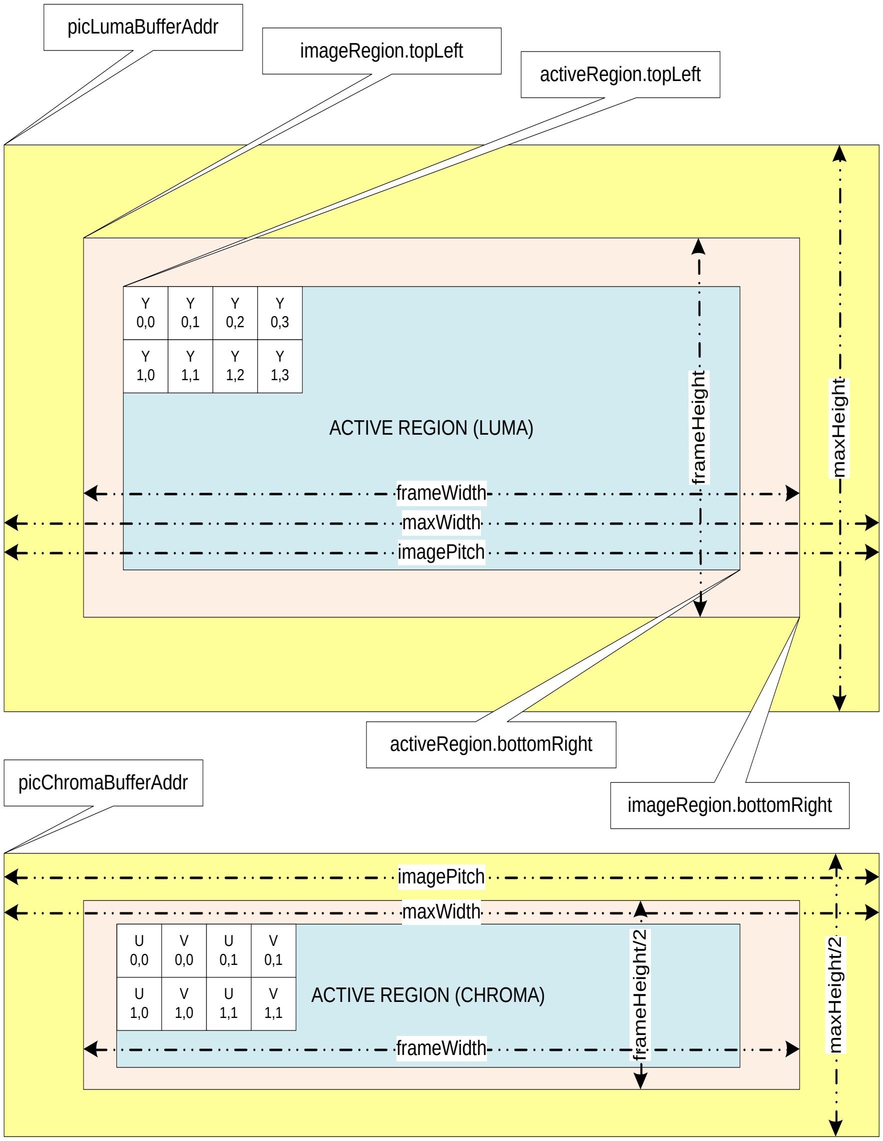
This Appendix explains picture format details for decoder. Decoder outputs YUV frames in formats specified in Table 1-1.

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



ActiveRegion and ImageRegion offsets for chroma are derived from luma offset
ChromaXoffset = lumaX_offset & 0xfffffffe;
ChromaYoffset = (lumaY_offset>>1) & 0xfffffffe;

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.* The dimensions of the chroma buffer region would as follows for different chroma formats:
    - YUV420 Semi Planar (NV12): *maxWidth x (maxHeight/2)*
    - YUV444 Planar: *maxWidth x maxHeight* (two such chroma buffers are needed: one each for Cb and Cr)
    - YUV422 YUYV: Single buffer for both luma and chroma data of size: *(2 x maxWidth) x maxHeight*
    - YUV420 Planar: (*maxWidth/2) x (maxHeight/2)* (two such chroma buffers are needed: one each for Cb and Cr)
    - YUV422 Planar: *maxWidth x (maxHeight/2)* (two such chroma buffers are needed: one each for Cb and Cr)
- *activeRegion*
    - The displayable region after cropping done by application.
- *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 padded regions and extra region due to alignment constraints.
    - 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*
    - No padding is done

6.3 Constraints on Buffer Allocation for Decoder

- ➢ *maxWidth* and *maxHeight* are inputs given by the decoder to 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 supported are as below

    - Minimum frameWidth = 32
    - Minimum frameHeight = 32
    - Maximum frameWidth = 4096
    - Maximum frameHeight = 4096

Chapter 7

Debug Trace Usage

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

7.1 Introduction

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

The primary use of Debug Trace Usage 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 JPEG decoder create time parameters are modified as

```
typedef struct IJPEGVDEC_Params{

  IVIDDEC3_Params viddecParams;

  XDAS_Int32     ErrorConcealmentON;

  XDAS_UInt32   debugTraceLevel;

  XDAS_UInt32   lastNFramesToLog;

  XDAS_UInt32   sliceSwitchON;

  XDAS_UInt32   numSwitchPerFrame;
  XDAS_UInt32   numRestartMarkerPerSwitch;
} IJPEGVDEC_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 IJPEGVDEC_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;
} IJPEGVDEC_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

Data Sync API Usage

This section explains the sub-frame level data synchronization API usage for MJPEG decoder from application point of view.

8.1 Description

Most of the TI Video Codec interfaces prior to IVIDENC2 and IVIDDEC3 allow frame level data communication capabilities. A user can configure the codec to encode/decode a complete frame but not any sub-frame level data communications. If at all any, then it is via codec's extended interface.

This document explains the sub-frame level data communication capabilities of video codec using data synchronization call backs defined with IVIDDEC3 interface.

8.2 MJPEG Decoder Input with Sub-frame Level Synchronization

This section explains the IVIDDEC3 interface details, which help to achieve the sub-frame level communications.

Table 8-1 and Table 8-2 explain the creation, control and handshake parameters related to sub frame level data communication for input data of MJPEG Decoder respectively.

"Details" column is a generic column and "valid values" column is specific to MJPEG Decoder input.

Table 8-1 Creation time parameter related to sub frame level data communication for input-data of MJPEG decoder

Parameter Name	Details	Valid values	
`IVIDDEC3_Params::inputDataMode`	Defines the mode of accepting the input data.	`IVIDEO_ENTIREFRAME`	Entire frame bit-stream is provided to the decoder
		`IVIDEO_FIXEDLENGTH`	Bitstream is provided to decoder after a fixed length of bytes. The length has to be multiple of 8K.
		`IVIDEO_SLICEMODE`	Bitstream is provided to decoder after having a single(or more) number of slice NAL Units.

`IVIDDEC3_Params::numInputDataUnits`	Unit of input data	*Don't care*

Table 8-2 Dynamic Parameters Related to sub–frame Level Data Communication for Input Data of MJPEG Decoder

Parameter Name	**Details**	**Valid values**
`IVIDDEC3_DynamicParams::getDataFxn`	This function is provided by the app/framework to the MJPEG Decoder. The decoder calls this function to get partial compressed bit-stream data from the app/framework. Apps/frameworks that don't support datasync should set this to NULL.	Any non-NULL value if `outputDataMode != IVIDEO_ENTIREFRAME`
`IVIDDEC3_DynamicParams::getDataHandle`	It defines the handle to be used while requesting data to application. This is a handle which the codec must provide when calling getDataFxn. For an algorithm, this handle is read-only; it must not be modified when calling the app-registered `IVIDDEC3_DynamicParams.getDataFxn()`. The app/framework can use this handle to differentiate callbacks from different algorithms.	Any Value

8.2.1 For Input mode equal to IVIDEO_SLICEMODE

In case of `inputDataMode = IVIDEO_SLICEMODE`, following points should be noted.

- No data is assumed to be available during process call, hence `IVIDDEC3_InArgs::numBytes` is not considered (it can be any non-zero positive value). All the data has to be provided via data synchronization calls.
- Application can provide maximum 32 non-contiguous buffers of varying size, but total size of data in one transaction has to be >= 8K bytes
- If the data provided during any data synch transaction is less than 8192 then decoder assumes it as end of frame.
- At the end of process call `IVIDDEC3_OutArgs::bytesConsumed` indicates the sum of total bytes consumed by decoder.

Refer Table 8-3 for the details of parameters being consumed by decoder during data synchronization transaction for `inputDataMode = IVIDEO_SLICEMODE.`

Table 8-3 Handshake parameters related to sub frame level data communication for input data of MJPEG decoder (inputDataMode = IVIDEO_SLICEMODE)

Parameter Name	Details	Valid values
`XDM_DataSyncDesc::size`	Size of the `XDM_DataSyncDesc` structure	`sizeof(XDM_DataSyncDesc)`
`XDM_DataSyncDesc::scatteredBlocksFlag`	Flag indicating whether the individual data blocks may be scattered in memory. Note that each individual block must be physically contiguous. Valid values `are XDAS_TRUE` and `XDAS_FALSE`. If set to `XDAS_FALSE`, the baseAddr field points directly to the start of the first block, and is not treated as a pointer to an array. If set to `XDAS_TRUE`, the baseAddr array must contain the base address of each individual block.	Flag indicating whether the individual data slices may be scattered in memory `(XDAS_TRUE` or `XDAS_FALSE)`.
`XDM_DataSyncDesc::baseAddr`	Base address of single data block or pointer to an array of data block addresses of size numBlocks. If `scatteredBlocksFlag` is set to `XDAS_FALSE`, this field points directly to the start of the first block, and is not treated as a pointer to an array. If `scatteredBlocksFlag` is set to `XDAS_TRUE`, this field points to an array of pointers to the data blocks.	If scatteredBlocksFlag is set to `XDAS_FALSE`, this field points directly to the start of the first block, and is not treated as a pointer to an array. If scatteredBlocksFlag is set to `XDAS_TRUE`, this field points to an array of pointers to the data blocks.
`XDM_DataSyncDesc::numBlocks`	Number of data blocks	***Constraint:*** *App can provide maximum 32 blocks in one transaction.* *1 <= numBlocks <= 32*
`XDM_DataSyncDesc::varBlockSizeFlag`	Flag indicating whether any of the data blocks vary in size.	`XDAS_TRUE` or `XDAS_FALSE` (slice sizes are not constant most of the time)
`XDM_DataSyncDesc::blockSizes`	Variable block sizes array.	If varBlockSizesFlag is `XDAS_TRUE`, this array contains the sizes of each slice. So Total_size = sum of (`blockSizes[0]` to `blockSizes[numBlocks -1]`. If varBlockSizesFlag is `XDAS_FALSE`, this contains the size of same-size slices. So Total_size = (numBlocks * blocSizes[0]). ***Constraint:*** *Total_size >= 8KB otherwise decoder assumes end of frame.*

8.2.2 For Input mode equal to IVIDEO_FIXEDLENGTH

In case of `inputDataMode = IVIDEO_FIXEDLENGTH`, following points should be noticed.

- No data is assumed to be available during process call, hence `IVIDDEC3_InArgs::numBytes` is is not considered (it can be any non-zero positive value). All the data has to be provided via data synch calls.
- Application can provide maximum one buffers of size as multiple of 8K during any data synch transaction.
    - During first data synch transaction, the data provided need not be multiple of 8KB.
    - If the data provided during any data synch transaction is less than 8KB then decoder assumes it as end of frame.
- At the end of process call `IVIDDEC3_OutArgs::bytesConsumed` indicates the sum of total bytes consumed by decoder.

Refer Table 8-4 for the details of parameters being consumed by decoder during data synch transaction for `inputDataMode = IVIDEO_FIXEDLENGTH`.

Table 8-4 Handshake parameters related to sub frame level data communication for input data of MJPEG decoder (inputDataMode = IVIDEO_FIXEDLENGTH)

Parameter Name	Details	Valid values
`XDM_DataSyncDesc::size`	Size of `the XDM_DataSyncDesc` structure	*`sizeof(XDM_DataSyncDesc)`*
`XDM_DataSyncDesc::scatteredBlocksFlag`	Flag indicating whether the individual data blocks may be scattered in memory. Note that each individual block must be physically contiguous. Valid values `are XDAS_TRUE` and `XDAS_FALSE`. If set to `XDAS_FALSE`, the baseAddr field points directly to the start of the first block, and is not treated as a pointer to an array. If set to `XDAS_TRUE`, the baseAddr array must contain the base address of each individual block.	Should be set to `XDAS_FALSE`.
`XDM_DataSyncDesc::baseAddr`	Base address of single data block or pointer to an array of data block addresses of size numBlocks. If scatteredBlocksFlag is set to `XDAS_FALSE`, this field points directly to the start of the first block, and is not treated as a pointer to an array.	This field points directly to the start of the data.

	If scatteredBlocksFlag is set to `XDAS_TRUE`, this field points to an array of pointers to the data blocks.	
`XDM_DataSyncDesc::numBlocks`	Number of data blocks	**Constraint** : App can provide maximum 1 block in one transaction.
`XDM_DataSyncDesc::varBlocksSizeFlag`	Flag indicating whether any of the data blocks vary in size.	Don't care assumed to be `XDAS_FALSE`
`XDM_DataSyncDesc::blockSizes`	Variable block sizes array.	Total_size = blockSizes[0]; **Constraint:** Except for first transaction, in rest all the transactions Total_size should be multiple of 8K bytes. If not decoder assumes it end of frame.

If application wants to use MJPEG Decoder to operate with sub-frame on input side:

- ❑ It should create the MJPEG Decoder with `IVIDDEC3_Params::inputDataMode = IVIDEO_SLICEMODE` or `IVIDEO_FIXEDLENGTH`.
- ❑ It should also make a control call with `IVIDDEC3_DynamicParams::getDataFxn` = non-NULL; to use sub frame level data communication, control call is mandatory.
- ❑ It should not provide the base address and available data of the input buffer during process call.
- ❑ `IVIDDEC3_DynamicParams::putDataFxn` == NULL && `IVIDDEC3_Params::inputDataMode != IVIDEO_ENTIREFRAME` is an erroneous situation and codec returns error during process call.

8.3 MJPEG Decoder Output with Sub-frame Level Synchronization

This section explains the IVIDDEC3 interface details, which helps to achieve the sub-frame level data synchronization for output.

Table 8-5, Table 8-6 and Table 8-7 explain the creation, control and handshake parameters related to sub-frame level data communication for output data of MJPEG Decoder respectively.

"Details" column is a generic column and "valid values" column is specific to MJPEG Decoder output.

Table 8-5 Creation time parameter related to sub frame level data communication for output data of MJPEG decoder

Parameter Name	**Details**	**Valid values**	
`IVIDDEC3_Params::outputDataMode`	Defines the mode of producing the output frame.	`IVIDEO_ENTIREFRAME`	Entire frame data is produced by decoder for display.
		`IVIDEO_NUMROWS`	Frame data is given in unit of Number of mb rows, each mb row is 16 lines of video.
`IVIDDEC3_Params::numOutputDataUnits`	Unit of output data	Don't care if `IVIDDEC3_Params::outputDataMode == IVIDEO_ENTIREFRAME` If `IVIDDEC3_Params::outputDataMode == IVIDEO_NUMROWS` then it defines the frequency at which decoder should inform to application about data availability. For example `numOutputDataUnits` = 2 means that after every 2 MB row (2*16 lines) availability in display buffer, decoder should inform to application.	

Table 8-6 Dynamic parameters related to sub frame level data communication for output data of MJPEG decoder

Parameter Name	**Details**	**Valid values**
`IVIDDEC3_DynamicParams::putDataFxn`	This function pointer is provided by the app/framework to the MJPEG Decoder. The decoder calls this function when sub-frame data has been put into an output buffer and is available.	Any non-NULL value if `outputDataMode != IVIDEO_ENTIREFRAME`
`IVIDDEC3_DynamicParams::putDataHandle`	It defines the handle to be used while informing data availability to application. This is a handle which the codec must provide when calling the app-registered. For an algorithm, this handle is read-only; it must not be modified when calling the app-registered `IVIDDEC3_DynamicParams.putDataFxn().` The app/framework can use this handle to differentiate callbacks from different algorithms.	Any Value

Table 8-7 Handshake parameters related to sub frame level data communication for output data of MJPEG decoder

Parameter Name	Details	Valid values
`XDM_DataSyncDesc::size`	Size of the `XDM_DataSyncDesc` structure	`Sizeof(XDM_DataSyncDesc)`
`XDM_DataSyncDesc::scatteredBlocksFlag`	Flag indicating whether the individual data blocks may be scattered in memory. Note that each individual block must be physically contiguous. Valid values are `XDAS_TRUE` and `XDAS_FALSE`. If set to `XDAS_FALSE`, the baseAddr field points directly to the start of the first block, and is not treated as a pointer to an array. If set to `XDAS_TRUE`, the baseAddr array must contain the base address of each individual block.	Don't care, always assumed to be `XDAS_FALSE`.
`XDM_DataSyncDesc::baseAddr`	Base address of single data block or pointer to an array of data block addresses of size numBlocks. If scatteredBlocksFlag is set to `XDAS_FALSE`, this field points directly to the start of the first block, and is not treated as a pointer to an array. If scatteredBlocksFlag is set to `XDAS_TRUE`, this field points to an array of pointers to the data blocks.	Don't care. Set to `XDAS_FALSE`.
`XDM_DataSyncDesc::numBlocks`	Number of data blocks	Number of rows given out by decoder in this call of putDataFxn. Value can be k*numOutputDataUnits. k = 1, 2 etc. Also, towards the end of frame, it will take value = [(no of rows in picture) mod (numOutputDataUnits)].
`XDM_DataSyncDesc::varBlockSizeFlag`	Flag indicating whether any of the data blocks vary in size.	**Don't care**, as unit of size is one row.
`XDM_DataSyncDesc::blockSizes`	Variable block sizes array.	**Don't care** since unit is assumed to be multiple of number of rows which is indicated by numBlocks.

If application wants to use MJPEG Decoder to operate with sub frame on output side:

- It should create the MJPEG decoder with `IVIDDEC3_Params::outputDataMode = IVIDEO_NUMROWS`.

- It should also make a control call with `IVIDDEC3_DynamicParams::putDataFxn = non-NULL`; to use sub frame level data communication, control call is mandatory.

- Address of the Luma and chroma output buffer will be present in decoded/display buffs. It will not be communicated via DataSyncDesc structure.

- If Video decode Media Controller thread doesn’t get scheduled before the next data availability, then in that situation codec give numBlocks as k*numOutputDataUnits.

  Constraint: Display order not being same as decode order with `IVIDDEC3_Params::outputDataMode = IVIDEO_NUMROWS,` is an erroneous situation.

- `IVIDDEC3_DynamicParams::putDataFxn == NULL && IVIDDEC3_Params::outputDataMode == IVIDEO_NUMROWS` is an erroneous situation and codec returns error during process call.

Chapter 9

Error Handling

This section explains the error handling by MJPEG 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`. Different error codes and their meanings are described below.

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. See below table for more details on error situations when flush can be performed.

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	`IJPEGDEC_ERR_UNSUPPORTED_VIDDEC3PARAMS`	This error code has been deprecated.	NA
1	`IJPEGDEC_ERR_UNSUPPORTED_VIDDEC3DYNAMICPARAMS`	Unsupported VIDDEC3DynamicParams are passed to the codec	Call GETSTATUS by passing extended Status structure to get more details about the error through extendedErrorCode0(/1/2/3) parameters.
2	`IJPEGDEC_ERR_UNSUPPORTED_JPEGDECDYNAMICPARAMS`	Unsupported extended class dynamic parameters are passed to the codec	Call GETSTATUS by passing extended Status structure to get more details about the error through extendedErrorCode0(/1/2/3) parameters.
3	`IJPEGDEC_ERR_NOSLICE`	Image does not have any slices and application is using slice level decoding	Disable slice level switching or provide an image with RST markers as input

4	`IJPEGDEC_ERR_MBDATA`	Invalid Input in MB data	If bytes available, advance BS pointer and pass fresh pointer
5	`IJPEGDEC_ERR_STANDBY`	HDVICP was not in standby when given to codec	Do HDVICP_Reset, XDM Reset and pass stream
6	`IJPEGDEC_ERR_INVALID_MBOX_MESSAGE`	Invalid MailBox Message has been received	Do HDVICP_Reset, XDM Reset and pass stream
7	`IJPEGDEC_ERR_HDVICP_RESET`	Hdvicp Reset Done is not proper	Do XDM Reset and pass stream.
16	`IJPEGDEC_ERR_HDVICP_WAIT_NOT_CLEAN_EXIT`	Hdvicp Wait exits early	Pass the next frame in the stream
17	`IJPEGDEC_ERR_FRAME_HDR`	Error in Frame header decoding	Pass the next frame in the stream
18	`IJPEGDEC_ERR_SCAN_HDR`	Error in Scan header decoding	Pass the next frame in the stream
19	`IJPEGDEC_ERR_HUFF_TBL_HDR`	Error in Huffman table decoding	Pass the next frame in the stream
20	`IJPEGDEC_ERR_QUANT_TBL_HDR`	Error in quant table decoding	Pass the next frame in the stream
21	`IJPEGDEC_ERR_OUTCHROMAFORMAT`	Not supported output chroma format set by the application to the codec	Refer to Table 1-1 for supported chroma formats
22	`IJPEGDEC_ERR_UNSUPPORTED_MARKER`	Unsupported Marker in the Input stream found	Pass the next frame in the stream
23	`IJPEGDEC_ERR_THUMBNAIL`	Error while decoding thumbnail marker	Pass the next frame in the stream
24	`IJPEGDEC_ERR_IRES_HANDLE`	Handle provided the Resource Manager is NULL.	Call delete and create again with proper handle
25	`IJPEGDEC_ERR_DYNAMIC_PARAMS_HANDLE`	Dynamic Params pointer passed to codec is NULL	Call delete and create again with proper handle
26	`IJPEGDEC_ERR_DATASYNC`	Data Sync Error	Pass the next frame in the stream
27	`IJPEGDEC_ERR_DOWNSAMPLE_INPUT_FORMAT`	Scaling/Downsampling has been enabled for unsupported chroma format combination	Decoder does not support scaling for this input or output chroma format
28	`IJPEGDEC_ERR_NOT_SUPPORTED_FEATURE`	Scaling/Downsampling/Thumbnail decoding has been enabled in Slice Level Decoding Mode	Decoder does not support scaling/downsampling or Thumbnail decoding of any chroma format in Slice Level decoding mode
29	`IJPEGDEC_ERR_NOT_SUPPORTED_RESOLUTION`	Unsupported resolution detected	Decoder does not support Width /Height less than 32 & Greater than 4096.

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

Bit	Error Code	Explanation	Recommended App Behaviour
0	`JPEG_DECODE_THUMBNAIL_ERROR`	Unsupported value passed to codec for 'decodeThumbnail' parameter	Call SETPARAMS with proper values set
1	`JPEG_DYNAMIC_PARAMS_HANDLE_ERROR`	Dynamic Params pointer passed to codec is NULL	Call SETPARAMS with a valid pointer
2	`JPEG_THUMBNAIL_MODE_ERROR`	Unsupported value passed to codec for 'thumbnailMode' parameter	Call SETPARAMS with proper values set
3	`JPEG_DOWNSAMPLING_FACTOR_ERROR`	Unsupported value passed to codec for 'downsamplingFactor ' parameter	Call SETPARAMS with proper values set
4	`JPEG_STREAMING_COMPLIANT_ERROR`	Unsupported value passed to codec for 'streamingCompliant ' parameter	Call SETPARAMS with proper values set
5	`JPEG_NON_INTERLEAVED_STREAMING_COMPLIANT_ERROR`	'streamingCompliant ' enabled for a non-interleaved image	Call SETPARAMS with proper values set – disable streamingCompliant for non interleaved images decoding
6	`JPEG_DECODE_HEADER_ERROR`	Unsupported value passed to codec for 'decodeHeader ' dynamic parameter	Call SETPARAMS with proper values set
7	`JPEG_DISPLAY_WIDTH_ERROR`	Unsupported value passed to codec for 'displayWidth ' dynamic parameter	Call SETPARAMS with proper values set
8	`JPEG_DYNAMIC_PARAMS_SIZE_ERROR`	Unsupported value passed to codec for 'size ' parameter of dynamic parmeters	Call SETPARAMS with proper values set
9	`JPEG_NULL_INSTANCE_HANDLE_ERROR`	Instance handle passed as NULL	Pass a valid handle
10	`JPEG_NULL_INARGS_POINTER_ERROR`	InArgs pointer passed as NULL in process call	Call process call with valid inArgs pointer to process call
11	`JPEG_NULL_OUTARGS_POINTER_ERROR`	OutArgs pointer passed as NULL in process call	Call process call with valid outArgs pointer to process call
12	`JPEG_NULL_INPUT_BUF_DESC_ERROR`	inbufdesc pointer passed as NULL in process call	Call process call with valid inbufdesc pointer to process call
13	`JPEG_NULL_OUTPUT_BUF_DESC_ERROR`	outbufdesc pointer passed as NULL in process call	Call process call with valid outbufdesc pointer to process call
14	`JPEG_INVALID_INARGS_SIZE`	Invalide 'size' parmeter for inArgs	Call process call with valid size for inArgs during process call

		passed in process call	
16	`JPEG_INVALID_OUTARGS_SIZE`	Invalide 'size' parameter for outArgs passed in process call	Call process call with valid size for inArgs during process call
17	`JPEG_NULL_INPUT_BUFFER_POINTER_ERROR`	Input buffer passed is NULL	Call process call with a valid Input buffer
18	`JPEG_NULL_OUTPUT_BUF_DESC_POINTER_ERROR`	pointer to outArgs->displaybufs passed is NULL	Call process call with a valid pointer for displaybufs
19	`JPEG_INVALID_NUM_OF_INPUT_BUFFERS_ERROR`	Invalid number of input buffers passed	Call process call with a valid value for number of input buffers
20	`JPEG_INVALID_INPUT_BYTES_ERROR`	Invalid input buffer size	Call process call with a valid input buffer size
21	`JPEG_INVALID_INPUT_BUFFER_MEMORY_TYPE_ERROR`	Unsupported memory region type for input buffer	Call process call with a valid memory region for input buffer
22	`JPEG_INVALID_NUM_OF_OUTPUT_BUFFERS_ERROR`	Invalid number of output buffers	Call process call with valid number of output buffers
23	`JPEG_NULL_OUTPUT_BUFFER_POINTER0_ERROR`	Output buffer -0 is passed as NULL to the codec	Call process call with a valid buffer pointer for output buffer-0
24	`JPEG_INVALID_OUTPUT_BUFFER0_SIZE_ERROR`	Output buffer -0 size is invalid	Call process call with a valid output buffer size
25	`JPEG_INVALID_OUTPUT_BUFFER0_MEMTYPE_ERROR`	Unsupported memory region passed for Output buffer -0	Call process call with a valid memory region for output buffer
26	`JPEG_NULL_OUTPUT_BUFFER_POINTER1_ERROR`	Output buffer -1 is passed as NULL to the codec	Call process call with a valid buffer pointer for output buffer-1
27	`JPEG_INVALID_OUTPUT_BUFFER1_SIZE_ERROR`	Output buffer -1 size is invalid	Call process call with a valid output buffer size
28	`JPEG_INVALID_OUTPUT_BUFFER1_MEMTYPE_ERROR`	Unsupported memory region passed for Output buffer -1	Call process call with a valid memory region for output buffer
29	`JPEG_NULL_OUTPUT_BUFFER_POINTER2_ERROR`	Output buffer -2 is passed as NULL to the codec	Call process call with a valid buffer pointer for output buffer-2
30	`JPEG_INVALID_OUTPUT_BUFFER2_SIZE_ERROR`	Output buffer -2 size is invalid	Call process call with a valid output buffer size
31	`JPEG_INVALID_OUTPUT_BUFFER2_MEMTYPE_ERROR`	Unsupported memory region passed for Output buffer -2	Call process call with a valid memory region for output buffer
32	`JPEG_INVALID_INPUT_ID_ERROR`	Invalid inputID passed to process call	Call process call with a valid inputID
33	`JPEG_NUM_VDMA_DESC_EXCEEDS_ERROR`	Error in VDMA open	Call HDVICP_Reset and pass the stream to process call
34	`JPEG_INVALID_SOI_MARKER_ERROR`	No start of image (SOI) maker found in the input stream	Pass the next frame in the stream
35	`JPEG_INVALID_MARKER_SEG_LENGTH_ERROR`	Invalid marker segment length	Pass the next frame in the stream
36	`JPEG_NON_STANDARD_MARKER_CODE_ERROR`	Marker Code is invalid	Pass the next frame in the stream

37	`JPEG_INVALID_QUANT_TABLE_TYPE_ERROR`	Number of Q tables in DQT is more than supported	Pass the next frame in the stream
38	`JPEG_QUANT_TABLE_BYTES_READ_ERROR`	Error in Q table reading	Pass the next frame in the stream
39	`JPEG_INVALID_HUFFMAN_TABLE_TYPE_ERROR`	Error in Huffman table reading	Pass the next frame in the stream
40	`JPEG_HUFFMAN_CODE_LENGTH_SIZE_EXCEED_ERROR`	Error in Huffman table code length	Pass the next frame in the stream
41	`JPEG_HUFFMAN_TABLE_MARKER_SEG_SIZE_ERROR`	Error in Huffman table marker syntax	Pass the next frame in the stream
42	`JPEG_HUFFMAN_TABLE_BYTES_READ_ERROR`	Error in Huffman table number of bytes to be read	Pass the next frame in the stream
43	`JPEG_INVALID_SAMPLE_PRECISION_ERROR`	Error in sample precision (only 8-bit samples are supported)	Pass the next frame in the stream
44	`JPEG_INVALID_NUM_COMPONENTS_ERROR`	Unsupported number of components in the header	Pass the next frame in the stream
45	`JPEG_FRAME_HDR_BYTES_READ_ERROR`	Error in frame header bytes	Pass the next frame in the stream
46	`JPEG_NOT_SUPPORTED_FORMAT_ERROR`	Unsupported chroma format	Pass the next frame in the stream
47	`JPEG_ARITHMETIC_DECODING_NOT_SUPPORTED_MARKER_ERROR`	Arithmetic decoding found, which is not supported	Pass the next frame in the stream
48	`JPEG_PROG_DECODING_NOT_SUPPORTED_MARKER_ERROR`	Arithmetic ext decoding found, which is not supported	Pass the next frame in the stream
49	`JPEG_LOSSLESS_DECODING_NOT_SUPPORTED_MARKER_ERROR`	Lossless decoding found, which is not supported	Pass the next frame in the stream
50	`JPEG_DIFFERENTIAL_DECODING_NOT_SUPPORTED_MARKER_ERROR`	Differential decoding found, which is not supported	Pass the next frame in the stream
51	`JPEG_JFIF_THUMBNAIL_IDENTIFIER_ERROR`	Error in JFIF identifier	Pass the next frame in the stream
52	`JPEG_JFIF_THUMBNAIL_BYTES_READ_ERROR`	Error in JFIF bytes	Pass the next frame in the stream
53	`JPEG_JFIF_EXTN_NO_SOI_ERROR`	SOI not found in JFIF extension	Pass the next frame in the stream
54	`JPEG_JFIF_NOT_SUPPORTED_FEATURE_ERROR`	Unsupported JFIF extension found	Pass the next frame in the stream
55	`JPEG_FORCECHROMA_OUTPUTCHROMA_FORMAT_MISMATCH_ERROR`	Unsupported force chroma format selected for the given input image	Call SETPARAMS with proper chroma format
56	`JPEG_INVALID_VERT_SCAN_FREQ_ERROR`	Error in vertical scan frequency for one of the components	Pass the next frame in the stream
57	`JPEG_INVALID_HORI_SCAN_FREQ_ERROR`	Error in horizontal scan frequency for one of the components	Pass the next frame in the stream
58	`JPEG_INVALID_QUANT_DEST_SELECTOR_ERROR`	Error in Q table ID for one of the components	Pass the next frame in the stream

59	`JPEG_DC_ENTROPY_CODING_DEST_ERROR`	Error in scan header parsing- DC component	Pass the next frame in the stream
60	`JPEG_AC_ENTROPY_CODING_DEST_ERROR`	Error in scan header parsing- AC component	Pass the next frame in the stream
61	`JPEG_ECD_VLD_OUT_OF_TABLE_ERROR`	ECD error: vld out of table	Pass the next frame in the stream
62	`JPEG_ECD_RESTART_INTERVAL_ERROR`	ECD error: invalid RST interval	Pass the next frame in the stream
63	`JPEG_ECD_BLOCK_COEFF_NUM_ERROR`	ECD error: invalid number of coefficients	Pass the next frame in the stream
64	`JPEG_GET_DATA_SYNC_NULL_FUNC_POINTER_ERROR`	Parameter ‘getDataFxn’ in dynamic params is NULL	Call SETPARAMS with a valid function pointer for getDataFxn
65	`JPEG_PUT_DATA_SYNC_NULL_FUNC_POINTER_ERROR`	Parameter ‘putDataFxn ‘ in dynamic params is NULL	Call SETPARAMS with a valid function pointer for putDataFxn
66	`JPEG_HDVICP_ACQUIRE_AND_CONFIGURE_ERROR`	Error in HDVICP acquire	Call HDVICP_Reset and pass the stream to process call
67	`JPEG_NULL_ALGORITHM_HANDLE_ERROR`	Algorithm handle provided is NULL	Call process call with a valid handle
68	`JPEG_GETVERSION_NULL_BUF_POINTER_ERROR`	Error in the buffer provided in GETVERSION through status->data	Call GETVERSION with proper buffer to hold version data
69	`JPEG_IRES_RESOURCE_DESC_ERROR`	resource descriptor pointer passed through IRES interface is NULL	Call algDelete and create the instance again
70	`JPEG_IRES_RESOURCE_DESC_HANDLE_ERROR`	handle to a resource passed through IRES interface is NULL	Call algDelete and create the instance again
71	`JPEG_NULL_STATUS_DATA_BUF`	NULL buffer passed through status->data.buf field for GETVERSION call	Call GETVERSION with proper buffer to hold version data
72	`JPEG_EXCEED_BYTES_CONSUMED_ERROR`	number of bytes consumed is more than total input bytes provided	Pass the next frame in stream
73	`JPEG_INPUT_DATASYNC_NUMBLOCKS_ERROR`	Unsupported number of blocks in input data sync	Pass a valid value for numBlocks in inputDataSyncParams (should be less than 32 and greater than 1)
74	`JPEG_INPUT_DATASYNC_BUFF_POINTER_ERROR`	Base address for input data sync provided is NULL	Pass a valid base address through input data sync
75	`JPEG_INPUT_DATASYNC_BLOCKSIZE_ERROR`	Block size provided through input data sync is zero	Pass a valid block size
76	`JPEG_INPUT_DATASYNC_NOT_VALID`	Unsupported combination of input data sync mode	Refer to datasync section in user guide for supported combinations
77	`JPEG_OUTPUT_DATASYNC_NUMBLOCKS_ERROR`	Unsupported number of blocks for	Pass a valid number of blocks

		output data sync call	
78	JPEG_SLICE_LEVEL_INPUT_NO_RST_MARKER_ERROR	No RST marker found for slice level input data sync	Pass the next frame in stream
79	JPEG_DOWNSAMPLING_IN_NON_TILED_ERROR	Scaling/Downsampling has been enabled when the output buffer provided to codec is not in TILED region	Provide output buffers to codec from TILED8 or TILED16 region
80	JPEG_DOWNSAMPLING_NOT_SUPPORTED_FORMAT_ERROR	Scaling/Downsampling has been enabled for unsupported chroma format combination	Decoder does not support scaling for this input or output chroma format
81	JPEG_DOWNSAMPLING_NOT_SUPPORTED_FEATURE_ERROR	Scaling/Downsampling has been enabled when data sync or slice level decoding is enabled	Decoder does not support scaling/downsampling feature when data sync or slice level decoding is enabled.
82	JPEG_THUMBNAIL_NOT_SUPPORTED_FEATURE_ERROR	Thumbnail mode has been enabled when data sync or slice level decoding is enabled	Decoder does not support Thumbnail decoding feature when data sync or slice level decoding is enabled.
83	JPEG_NOT_SUPPORTED_WIDTH_ERROR	Less than 32 of Minimum Width and Greater than 4096 of Maximum Width is enabled.	Decoder does not support the Width less than 32 and greater than 4096 for decoding .
84	JPEG_NOT_SUPPORTED_HEIGHT_ERROR	Less than 32 of Minimum Height and Greater than 4096 of Maximum Height is enabled.	Decoder does not support the Height less than 32 and greater than 4096 for decoding.

This page is intentionally left blank

Chapter 10

Slice Level Decoding

This section explains the support of Slice Level Decoding in MJPEG decoder.

10.1 Introduction

This section explains the overall design that has been adopted for slice level decoding.

The primary uses of Slice Level Decoding are:

1) In multi-instance scenario, context switching can happen at slice level leading to better performance in real-time.
2) For the error inputs, output will be visually very good compared to entire frame decoding.

Each switch can be considered as one process call, so once all the switches have been decoded, Output buffer will be freed.

In slice level decoding, input for different switches may not be contiguous in memory but output should be contiguous for a frame.

10.2 Enabling and using slice level decoding

The following three parameters in create time parameters will be used to configure slice level decoding.

1) sliceSwitchON
2) numSwitchPerFrame
3) numRestartMarkerPerSwitch

Hence, the JPEG decoder create time parameters are as follows:

```
typedef struct IJPEGVDEC_Params{
  IVIDDEC3_Params viddecParams;
  XDAS_Int32     ErrorConcealmentON;
  XDAS_UInt32   debugTraceLevel;
  XDAS_UInt32   lastNFramesToLog;
  XDAS_Int32     sliceSwitchON;
```

XDAS_UInt32 numSwitchPerFrame;

XDAS_UInt32 numRestartMarkerPerSwitch;

} IJPEGVDEC_Params;

10.2.1 sliceSwitchON

This parameter configures the codec to decode the input in slice mode.

- 0: Disables slice level decoding feature.
- 1: Enables the slice level decoding feature.

If “sliceSwitchON” parameter is “ENABLED” , slice level decoding of the input will be done depending on the following two modes :

1. numSwitchPerFrame
2. numRestartMarkerPerSwitch

10.2.2 numSwitchPerFrame

This parameter’s value (if non-zero) is valid only when “sliceSwitchON” parameter is ENABLED, when “sliceSwitchON” is disabled this parameter is not used.

There are two modes for decoding Input Image in slice mode:

In this mode, “numSwitchPerFrame” parameter tells us how many switches has to happen to decode one Frame.

This parameter has higher priority than “numRestartMarkerPerSwitch”, when both the modes “numRestartMarkerPerSwitch” and “numSwitchPerFrame” are non-zero , only “numSwitchPerFrame” parameter will be considered and “numRestartMarkerPerSwitch” mode will be discarded.

10.2.3 numRestartMarkerPerSwitch

This parameter’s value (if non-zero) is valid only when “sliceSwitchON” parameter is ENABLED , when “sliceSwitchON” is disabled this parameter is not used.

In this mode, “numRestartMarkerPerSwitch” parameter tells us how many slices to decode every switch. Codec has to calculate how may switches will be there and codec has to free the output buffer when all the slices in the Input has been decoded.

10.3 Requirements On The Application

The following are the requirements on the application side:

1. The application should be capable of configuring *sliceSwitchON* , *numSwitchPerFrame* and *numRestartMarkerPerSwitch* which are part of the Initialization Parameters of the codec

2. The application should be capable of handling the input according to every switch.
3. Each switch Input maybe independent of each other but it has to be in the same order. (If there are 3 switches in a Frame , application has to pass switch 1 first, switch 2, and then switch 3).
4. The application should not enable downsampling and thumbnail decoding along with slice level decoding.