MJPEG Encoder on HDVICP2 and Media Controller Based Platform

User's Guide

Literature Number: SPRUH27
June 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

Read This First

About This Manual

This document describes how to install and work with Texas Instruments' (TI) MJPEG Encoder 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 Encoder on HDVICP2 and Media Controller Based Platform.

- **Chapter 6 – Picture Format,** provides information on format of YUV buffers provided to encoder.
- **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 encoder from application point of view.
- **Chapter 9 – Error Handling,** explains the error handling and error robustness features of this MJPEG Encoder.

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.

Table 1-1 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
DMAN	DMA Manager
EVM	Evaluation Module
HDTV	High Definition Television
IRES	Interface standard to request and receive handles to resources
ISO	International Standards Organization
HDVICP2	Image Video Accelerator
MB	Macro Block
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 Encoder on HDVICP2) and version number. The version number of the codec is included in the title of the Release Notes that accompanies this codec.

Trademarks

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

All trademarks are the property of their respective owners.

This page is intentionally left blank

Contents

Figures

This page is intentionally left blank

Tables

Chapter 1

Introduction

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

Topic **Page**

1.1 Overview of XDAIS and XDM

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

1.1.1 XDAIS Overview

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

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

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

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

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

1.1.2 XDM Overview

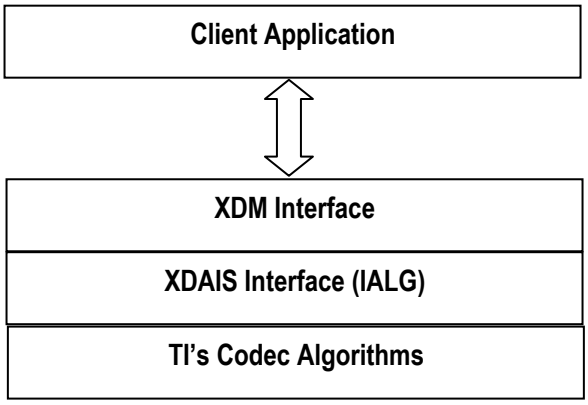
In the multimedia application space, you have the choice of integrating any codec into your multimedia system. For example, if you are building a video encoder system, you can use any of the available video encoders (such as MPEG4, H.263, or H.264) in your system. To enable easy integration with the 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 encoder, then you can easily replace MPEG4 with another XDM-compliant video encoder, 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).

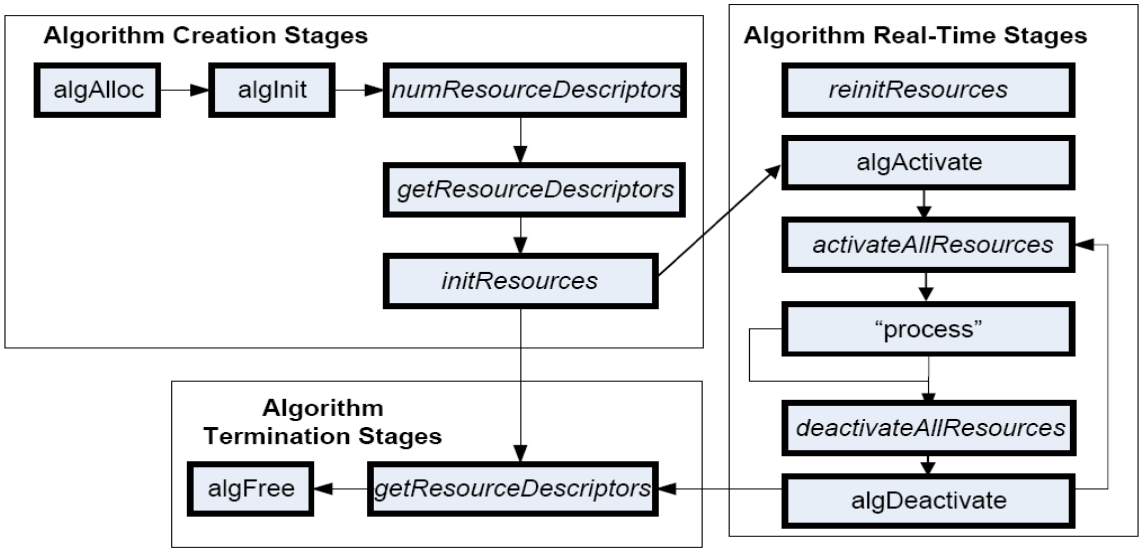


Figure 1-1 IRES Interface Definition and Function Calling Sequence

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

1.2 Overview of MJPEG Encoder

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 discrete cosine transform (DCT) and quantization 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 in JPEG:

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

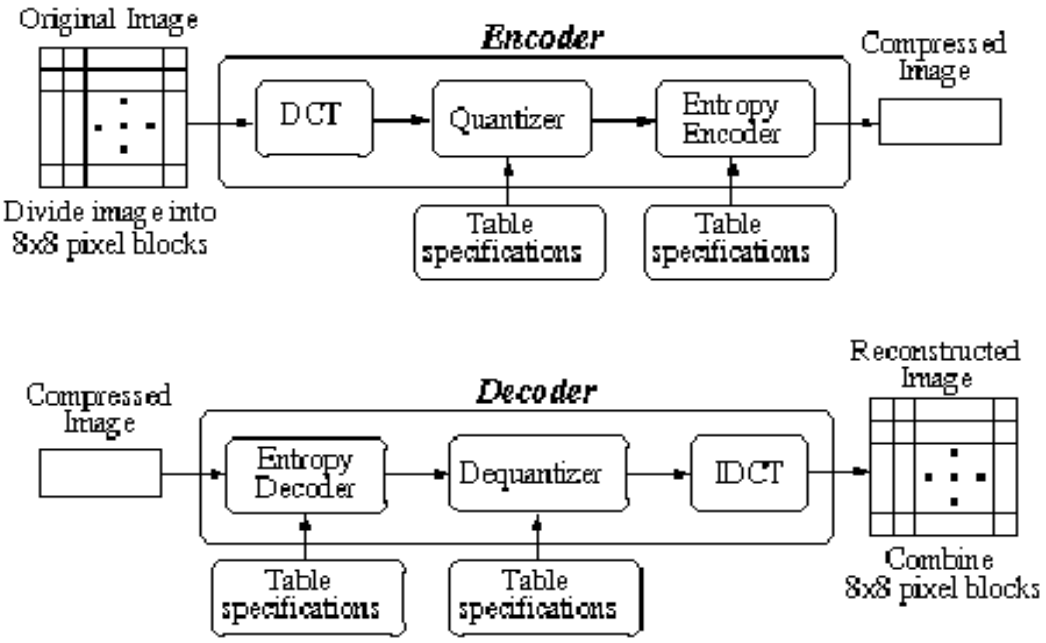


*Figure **Error! No text of specified style in document.**-2. Block diagram of Simple JPEG encoder-decoder*

From this point onwards, all references to MJPEG (Motion JPEG) Encoder means JPEG Baseline Sequential Encoder used in video mode (i.e. continuous frame encoding in JPEG format).

1.3 Supported Services and Features

This user guide accompanies TI's implementation of MJPEG Encoder on the IVA-HD platform.

This version of the codec has the following supported features:

- eXpressDSP Digital Media (XDM IVIDENC2) compliant
- Supports baseline sequential mode for interleaved data formats (single scan)
- Supports 8 bpp per component
- Supports YUV 444 Planar, YUV 422 YUYV IBE, YUV 420 Semi-Planar and Gray scale chroma formats for input
- Supports YUV 444, YUV 422, YUV 420 and Gray scale chroma formats for output (only interleaved formats are supported)
- Supports all resolutions up to 4320x4096

- Supports a maximum of three components
- Supports user-configurable encoding parameters
- Supports 16-bit quantization tables
- Supports selection of quality level by user
- Supports user-defined quantization tables
- Supports insertion of restart marker
- Supports insertion of JPEG File Interchange Format (JFIF) marker segment
- Supports insertion of comment marker segment
- Supports insertion of Exif marker segment
- Supports insertion of thumbnail in JFIF or Exif marker segment
- Supports sub-frame data synchronization for input and output
- Supports graceful exit under error conditions
- Supports multi-channel functionality
- Supports debug trace dump

Limitations:

- Does not support extended sequential mode
- Does not support 12 bits per sample
- Does not support non-interleaved YCbCr output (multiple scans)
- Does not support encoding of thumbnails. This encoder supports only *insertion* of encoded thumbnail data provided by the application.

This page is intentionally left blank

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 encoder has been developed using Code Composer Studio 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 encoder has been compiled, assembled, archived, and linked using the code generation tools version 4.5.1.

  Although CG Tools v4.5.1 is a part of Code Composer Studio v4 installation, it is recommended that you re-install CG tools after downloading from the following link.

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

- **HDVICP2 Simulator:** This encoder has been tested using HDVICP2 Simulator version 5.0.16 (HDVICP2 Simulation CSP 1.1.5). 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/IVAHD/site.xml

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

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

2.2 Installing the Component

The codec component is released as a compressed archive. To install the codec, extract the contents of the zip file onto your local hard disk. The zip file extraction creates a top-level directory called 500.V.MJPEG.E.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*.

Figure 2-1. Component Directory Structure

Table 2-1 provides a description of the sub-directories created in the 500.V.MJPEG.E.IVAHD.01.00 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 Encoder
\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, IVA-HD picture format docs
\inc	Contains interface header files of MJPEG encoder
\lib	Contains jpegenc_ti_host.lib – IVA-HD MJPEG Encoder 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 ivahd_sim_user_guide.pdf present in <CCSv4 Installation Dir>\simulation_csp_omap4\docs\pdf\ directory.

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

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

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

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

Set environment variable CG_TOOL_DIR to <cgtools_install_dir>.

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

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

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

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

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

The version of the XDC tools required is 3.20.04.68 GA.

2.3.1 Installing Framework Component (FC)

You can download FC from the TI website:

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

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

The test application uses the following IRES and XDM files:

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

2.3.2 Installing XDC Tools

XDC Tools is required to build the test application. The test application uses the standard files like <std.h> from XDC tools. This encoder 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 Encoder 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\jpegenc_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 \client\build\TestApp<DeviceName>\out\ sub-directory, select the codec executable "jpegenc_ti_testapp.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 M3_Video device and select **Target > Run** to execute the application.

8) Test application will take input streams from \client\test\testvecs\input\ directory and generates outputs in \client\test\testvecs\output\ directory.

2.4.3 Running the Sample Test Application on DM816x EVM

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

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

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

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

4) Select Cortex_M3_RTOS_0 device, right click and choose "Connect Target" and wait for emulator to connect to CortexM3.

5) Select Cortex_M3_RTOS_0 device and **Target > Load Program**, browse to \500.V.MJPEG.E.IVAHD.01.00\IVAHD_001\client\build\TestAppDM816x\out\ sub-directory, select the codec executable "jpegenc_ti_testapp.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 \client\test\testvecs\input\ directory and generates outputs in \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 (encoder_testvecs.cfg) - specifies input and output files for the sample test application.
- A sample Encoder configuration file named encoder_testparams.cfg – specifies the configuration parameters used by the test application to configure the Encoder.

2.5.1 Generic Configuration File

The sample test application shipped along with the codec uses the configuration file, encoder_testvecs.cfg for determining the input and output files for running the codec. The encoder_testvecs.cfg file is available in the \client\test\testvecs\config sub-directory.

The format of the encoder_testvecs.cfg file is:

```
Mode
Config
Input
Output or Reference
```

where:

- `Mode` may be set as:
  - 1 - for compliance checking.
  - 0 - for writing the output to the output file
- `Config` is the Encoder configuration file. For details, see Section 2.5.2.
- `Input` is the input file name (use complete path).
- `Output` is the output (.jpg) file name (output dump mode). `Reference` is the reference (.jpg) file name (in compliance checking mode).

A sample Testvecs.cfg file is as shown:

```
0
..\..\Test\TestVecs\Config\encoder_testparams.cfg
..\..\Test\TestVecs\Input\davincieffect_qcif_yuv420_sp.yuv
..\..\Test\TestVecs\Output\davincieffect_qcif_yuv420_sp.jpg
```

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

2.5.2 Encoder Configuration File

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

A sample encoder_testparams.cfg file is as shown:

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

###############################################################################
# Files
###############################################################################
NumInputUnits       = 1     # Number of units of input-data
                            # (ex. 3 rows to be encoded).

MaxWidth            = 1920  # Max Frame width should be multiple of 16
MaxHeight           = 1088  # Max Frame height should be multiple of 16
DataEndianess       = 1     # 1=> 8-bit Big Endian stream.
InputChromaFormat   = 9     # XDM_YUV_420SP format
InputContentType    = 0     # IVIDEO_PROGRESSIVE
OperatingMode       = 1     # Encode Mode
InputDataMode       = 3     # Process entire frame.
OutputDataMode      = 3     # Encode entire frame into a bitstream in
                            # single call.
NumOutputUnits      = 1     # Number of units of output-data
                            # (ex. 1 Slice/Frame encoded stream).



###############################################################################
# Encoder Control
###############################################################################

FrameWidth              = 176 # Frame width should be multiple of 16
FrameHeight             = 144 # Frame height should be multiple of 16

QualityFactor           = 20  # Quality Setting to be used.
                              # Ranges between 2 and 97.
                              # 2 => Lowest Quality. 97 => Best Quality.

CaptureWidth        = 176  # Image width to compute image pitch.
                           # If Capture Width is > Image Width then
                           # use the former for image pitch.
generateHeader      = 0    # Set 1 => Generate Header Only
debugTraceLevel     = 0    # Specifies the debug trace level
lastNFramesToLog    = 0    # Specifies the number of past frames
                           # to log debug trace. If debugTraceLevel is
                           # greater than 0, lastNFramesToLog must range
                           # between 0 and 10.
tilerEnable         = 1    # 1 => Luma in TILER8, 0=> Luma in Raw.
chromaTilerMode     = 0    # Valid only if tilerEnable = 1
                           # 1 => Chroma in TILER8, 0 => Chroma in
                           # TILER16
```

Any field in the `IVIDENC2_Params` or `IVIDENC2_DynamicParams` structure can be set in the encoder_testparams.cfg file using the syntax shown above. If you specify additional fields in the encoder_testparams.cfg file, ensure to modify the test application appropriately to handle these fields.

Note:

Chroma formats supported in this release are YUV420 semi-planar, YUV444 planar, YUV422 YUYV IBE, and YUV400.

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.

Topic	Page

3.1 Overview of the Test Application

The test application exercises the `IVIDENC2` base class of the MJPEG Encoder library. The main test application files are jpegenc_ti_Test.c and jpegenc_ti_rman_config.c. These files are available in the \enc\jpeg\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.

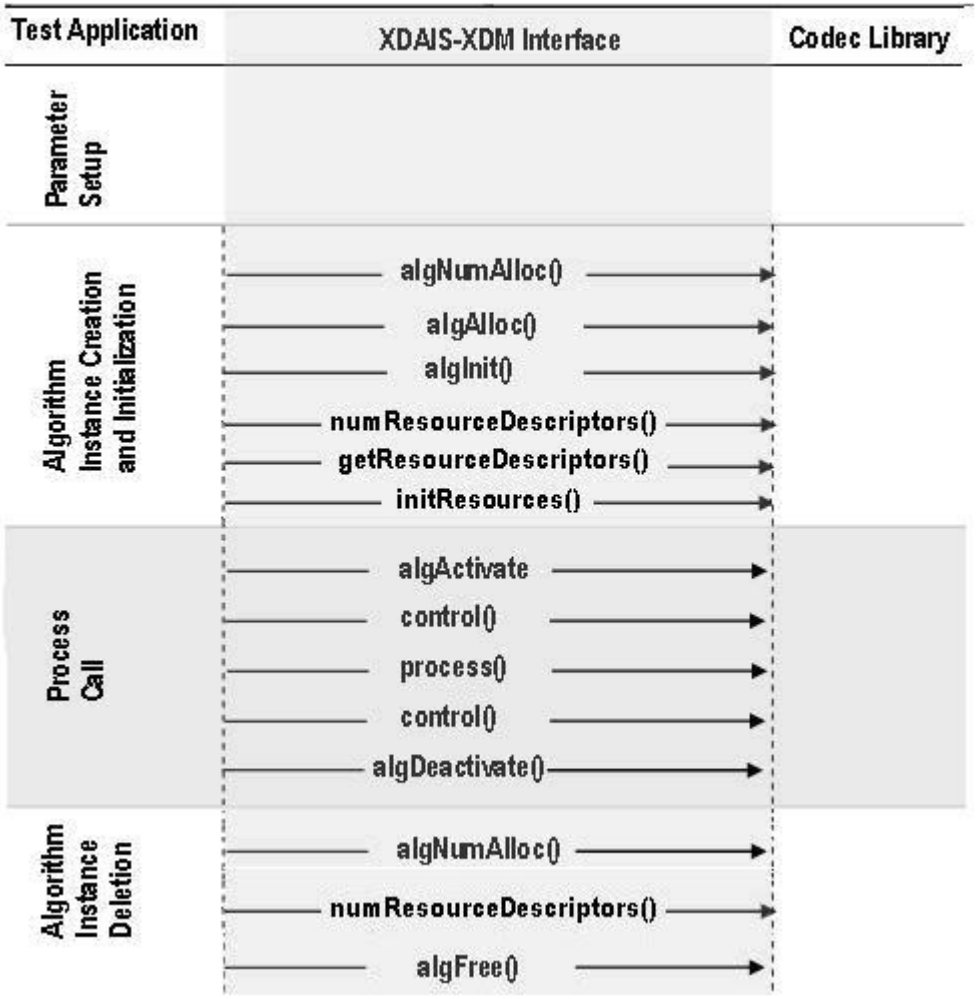


Figure 3-1. Test Application Sample Implementation

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 Encoder configuration files.

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

1) Opens the encoder configuration file, (encoder_testparams.cfg) and reads the various configuration parameters required for the algorithm. For more details on the configuration files, see Section 2.4.2.

2) Sets the `IVIDENC2_Params` structure based on the values it reads from the encoder_testparams.cfg file.

3) 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:

- Encoder 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 `IVIDENC2_InArgs` and `IVIDENC2_OutArgs` structures.

4) On the call to the `process()` function for encoding/decoding a single frame of data, the software triggers the start of encode/decode. After triggering the start of the encode/decode frame, the video task can be put to `SEM-pend` state using semaphores. On receipt of interrupt signal at the end of frame encode/decode, the application releases the semaphore and resume the video task, which does any book-keeping operations by the codec and updates the output parameter of `IVIDENC2_OutArgs` structure.

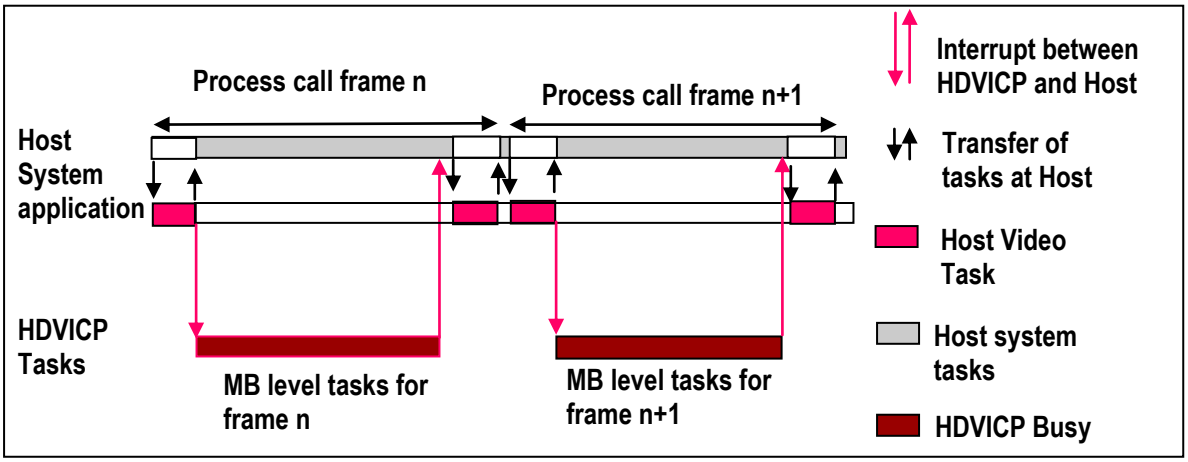


Figure 3-2. Process call with Host release

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 Encoder 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.

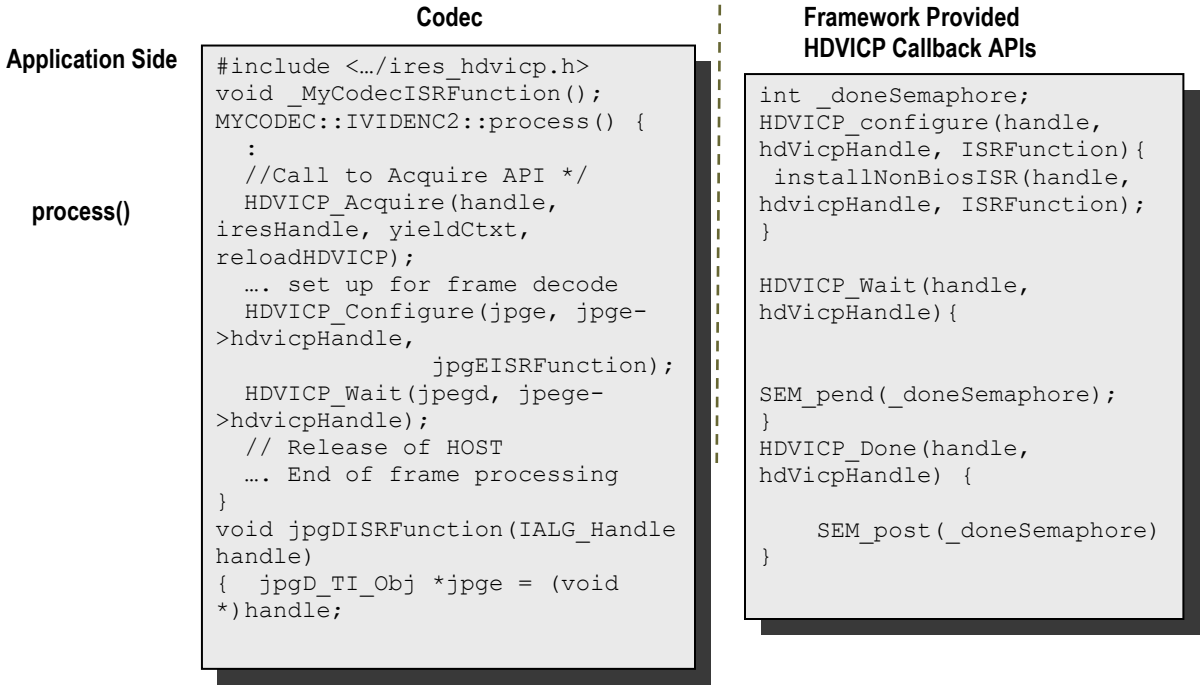


Figure 3-3. Interaction Between Application and Codec

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 Media Controller and IVA-HD(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 `IVIDENC2` base class of the MJPEG Encoder.

Table 3-1 Process() Implementation

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

  TestApp_SetInitParams(&params.videncParams);

  /*---------------- Encoder creation -----------------*/
  handle = (IALG_Handle) JPEGVENC_create();

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


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

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

  /*-----------------------------------------------------*/
  /* Start the process : To start encoding a frame      */
  /*-----------------------------------------------------*/
      retVal = JPEGVENC_encodeFrame
            (
              handle,
              (XDM1_BufDesc *)&inputBufDesc,
              (XDM_BufDesc *)&outputBufDesc,
              (IVIDENC2_InArgs *)&inArgs,
              (IVIDENC2_OutArgs *)&outArgs
            );

     /* Get the status of the encoder using control   */
     JPEGVENC_control(handle, XDM_GETSTATUS);

     /* Get Buffer information :                      */
     JPEGVENC_control(handle, XDM_GETBUFINFO);
}  while(1);
/* end of Do-While loop - which encodes 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.

Topic **Page**

4.1 Symbolic Constants and Enumerated Data Types

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

Table 4-1 List of Enumerated Data Types

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

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

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_SKIP_IB	Skip I and B frame/field(s). Not applicable for MJPEG encoder.
	IVIDEO_SKIP_PB	Skip P and B frame/field(s). Not applicable for MJPEG encoder.
	IVIDEO_SKIP_IPB	Skip I/P/B/BI frames Not applicable for MJPEG encoder.
	IVIDEO_SKIP_IDR	Skip IDR Frame Not applicable for MJPEG encoder.
	IVIDEO_SKIP_NONREFERENCE	Skip non reference frame Not applicable for MJPEG encoder.
	IVIDEO_SKIP_DEFAULT	Not applicable for MJPEG encoder.
IVIDEO_VideoLayout	IVIDEO_FIELD_INTERLEAVED	Buffer layout is interleaved. Not applicable for MJPEG encoder.
	IVIDEO_FIELD_SEPARATED	Buffer layout is field separated. Not applicable for MJPEG encoder.
	IVIDEO_TOP_ONLY	Buffer contains only top field. Not applicable for MJPEG encoder.
	IVIDEO_BOTTOM_ONLY	Buffer contains only bottom field. Not applicable for MJPEG encoder.
IVIDEO_OperatingMode	IVIDEO_DECODE_ONLY	Decoding Mode. Not supported.
	IVIDEO_ENCODE_ONLY	Encoding Mode.
	IVIDEO_TRANSCODE_FRAMELEVEL	Transcode Mode of operation (encode/decode), which consumes /generates transcode information at the frame level. Not supported.
	IVIDEO_TRANSCODE_MBLEVEL	Transcode Mode of operation (encode/decode), which consumes /generates transcode information at the MB level. Not supported.
	IVIDEO_TRANSRATE_FRAMELEVEL	Transrate Mode of operation for encoder, which consumes transrate information at the frame level. Not supported.
	IVIDEO_TRANSRATE_MBLEVEL	Transrate Mode of operation for encoder, which consumes transrate information at the MB level. Not supported.
IVIDEO_OutputFrameStatus	IVIDEO_FRAME_NOERROR	Output buffer is available. Not applicable for encoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_FRAME_NOTAVAILABLE	Codec does not have any output buffers. Not applicable for encoder.
	IVIDEO_FRAME_ERROR	Output buffer is available and corrupted. Not applicable for encoder.
	IVIDEO_FRAME_OUTPUTSKIP	The video frame was skipped (that is not decoded). Not applicable for encoder.
	IVIDEO_OUTPUTFRAMESTATUS_DEFAULT	Default set to IVIDEO_FRAME_NOERROR. Not applicable for encoder.
IVIDEO_PictureType	IVIDEO_NA_PICTURE	Frame type not available. Not applicable for MJPEG encoder.
	IVIDEO_I_PICTURE	Intra coded picture. Not applicable for MJPEG encoder.
	IVIDEO_P_PICTURE	Forward inter coded picture. Not applicable for MJPEG encoder.
	IVIDEO_B_PICTURE	Bi-directional inter coded picture. Not applicable for MJPEG encoder.
IVIDEO_DataMode	IVIDEO_FIXEDLENGTH	Output of the encoder is in multiples of a fixed length (example, 4K) (output side for encoder).
	IVIDEO_SLICEMODE	Slice mode of operation (Output side for encoder).
	IVIDEO_NUMROWS	Number of MCU rows (Input side for encoder).
	IVIDEO_ENTIREFRAME	Processing of entire frame data
XDM_DataFormat	XDM_BYTE	Big endian stream (default value)
	XDM_LE_16	16-bit little endian stream. Not supported in this version of MJPEG Encoder.
	XDM_LE_32	32-bit little endian stream. Not supported in this version of MJPEG Encoder.
	XDM_LE_64	64-bit little endian stream. Not supported in this version of MJPEG Encoder.
	XDM_BE_16	16-bit big endian stream. Not supported in this version of MJPEG Encoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_BE_32	32-bit big endian stream. Not supported in this version of MJPEG Encoder.
	XDM_BE_64	64-bit big endian stream. Not supported in this version of MJPEG Encoder.
XDM_ChromaFormat	XDM_YUV_420P	YUV 4:2:0 planar. Not supported in this version of MJPEG Encoder.
	XDM_YUV_422P	YUV 4:2:2 planar. Not supported in this version of MJPEG Encoder.
	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 Encoder.
	XDM_YUV_444P	YUV 4:4:4 planar.
	XDM_YUV_411P	YUV 4:1:1 planar. Not supported in this version of MJPEG Encoder.
	XDM_GRAY	Gray format.
	XDM_RGB	RGB color format. Not supported in this version of MJPEG Encoder.
	XDM_YUV_420SP	YUV 4:2:0 chroma semi-planar
	XDM_ARGB8888	ARGB8888 color format. Not supported in this version of MJPEG Encoder.
	XDM_RGB555	RGB555 color format. Not supported in this version of MJPEG Encoder.
	XDM_RGB565	RGB565 color format. Not supported in this version of MJPEG Encoder.
	XDM_YUV_444ILE	YUV 4:4:4 interleaved (little endian) color format. Not supported in this version of MJPEG Encoder.
XDM_MemoryType	XDM_MEMTYPE_ROW	Raw Memory Type (deprecated)
	XDM_MEMTYPE_RAW	Raw Memory Type i.e., Linear (standard) memory.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_MEMTYPE_TILED8	2D memory in 8-bit container of tiled memory space.
	XDM_MEMTYPE_TILED16	2D memory in 16-bit container of tiled memory space.
	XDM_MEMTYPE_TILED32	2D memory in 32-bit container of tiled memory space. Not supported in this version of MJPEG Encoder.
	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. Not supported in this version of MJPEG Encoder.
	XDM_GETBUFINFO	Query algorithm instance regarding the properties of input and output buffers
	XDM_GETVERSION	Query the algorithm's version. The result will be returned in the data field of the Status structure. Application has to allocate memory for a buffer passed through data field. The minimum buffer size required is 96 bytes.
	XDM_GETCONTEXTINFO	Query a split codec part for its context needs. Not supported in this version of MJPEG Encoder.
	XDM_GETDYNPARAMSDEFAULT	Query algorithm instance regarding the dynamic parameters default values.
	XDM_SETLATEACQUIREARG	Set an algorithm's 'late acquire' argument.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
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
	XDM_FATALERROR	Bit 15 1 - Fatal error 0 - Recoverable error
IJPEGVENC_ExtendedErrorCodes	IJPEGVENC_ERR_UNSUPPORTED_VIDENC2PARAMS	Bit 0 This error code has been deprecated.
	IJPEGVENC_ERR_UNSUPPORTED_VIDENC2DYNAMICPARAMS	Bit 1 1 - Unsupported VIDENC2DynamicParams have been passed to the codec 0 - Error not found
	IJPEGVENC_ERR_UNSUPPORTED_JPEGENCDYNAMICPARAMS	Bit 2 1 - Unsupported JPEGVENC_DynamicParams (i.e., extended) have been passed to the codec 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IJPEGVENC_ERR_IMPROPER_DATASYNC_SETTING	Bit 3 This error code has been deprecated.
	IJPEGVENC_ERR_NOSLICE	Bit 4 This error code has been deprecated.
	IJPEGVENC_ERR_SLICEHDR	Bit 5 This error code has been deprecated.
	IJPEGVENC_ERR_MBDATA	Bit 6 This error code has been deprecated.
	IJPEGVENC_ERR_UNSUPPFEATURE	Bit 7 This error code has been deprecated.
	IJPEGVENC_ERR_STREAM_END	Bit 16 This error code has been deprecated.
	IJPEGVENC_ERR_INVALID_MBOX_MESSAGE	Bit 17 1 - Invalid MailBox Message has been received 0 - Error not found
	IJPEGVENC_ERR_HDVICP_RESET	Bit 18 This error code has been deprecated.
	IJPEGVENC_ERR_HDVICP_WAIT_NOT_CLEAN_EXIT	Bit 19 1 - Exit from HDVICP2 is not clean 0 - Error not found
	IJPEGVENC_ERR_IRES_RESHANDLE	Bit 20 This error code has been deprecated.
	IJPEGVENC_ERR_STANDBY	Bit 21 1 - HDVICP was not in standby when given to codec 0 - Error not found
	IJPEGVENC_ERR_INPUT_DATASYNC	Bit 22 1 - Error in the Input Data Sync Call Function 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IJPEGVENC_ERR_OUTPUT_DATASYNC	Bit 23 1 - Error in the Output Data Sync Call Function 0 - Error not found
IjpegVENC_ErrorStatus	JPEG_DYNAMIC_PARAMS_HANDLE_ERROR	Bit 0 1 - Dynamic Params pointer passed to codec is NULL 0 - Error not found
	JPEG_STATUS_HANDLE_ERROR	Bit 1 1 - Status Pointer passed to codec is NULL 0 - Error not found
	JPEG_DYNAMIC_PARAMS_SIZE_ERROR	Bit 2 1 - Invalid size of dynamic parameters passed to codec 0 - Error not found
	JPEG_ENCODE_HEADER_ERROR	Bit 3 1 - Invalid GenerateHeader value passed to the codec 0 - Error not found
	JPEG_UNSUPPORTED_RESOLUTION	Bit 4 1 - Frame height and Frame width passed to the codec is less than 32 or greater than Max Width and Max Height provided during create time 0 - Error not found
	JPEG_CAPTURE_WIDTH_ERROR	Bit 5 1 - Invalid Capture Width value passed to the codec 0 - Error not found
	JPEG_GET_DATA_FXN_NULL_POINTER	Bit 6 1 - No call back function pointer is passed for getDataFxn 0 - Error not found
	JPEG_GET_BUFFER_FXN_NULL_POINTER	Bit 7 1 - No call back function pointer is passed for getBufferFxn OR putDataFxn 0 - Error not found
	JPEG_INVALID_RESTART_INTERVAL_ERROR	Bit 8 1 - Invalid Restart Interval value (< 0) passed to the codec 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	JPEG_INVALID_QUALITY_FACTOR_ERROR	Bit 9 1 - Invalid Quality factor value passed to the codec. Valid range is [1, 100]. 0 - Error not found
	JPEG_INVALID_INPUT_CHROMA_FORMAT_ERROR	Bit 10 1 - Invalid chroma format passed to the codec 0 - Error not found
	JPEG_NULL_QUANT_TABLE_POINTER_ERROR	Bit 11 1 - Both quality factor and user defined quantization table are valid 0 - Error not found
	JPEG_NULL_INARGS_POINTER_ERROR	Bit 12 1 - InArgs Pointer passed to codec in process call is NULL 0 - Error not found
	JPEG_NULL_INARGS_APP_POINTER_ERROR	Bit 13 1 - Both APP0 & APP1 Segments are passed to the codec 0 - Error not found
	JPEG_INARGS_SIZE_ERROR	Bit 14 1 - Invalid size of InArgs passed to the codec 0 - Error not found
	JPEG_INVALID_INPUT_BYTES_ERROR	Bit 15 This error code has been deprecated.
	JPEG_INVALID_INPUT_ID_ERROR	Bit 16 1 - Value of 0 was passed as input ID 0 - Error not found
	JPEG_NULL_INPUT_BUF_DESC_ERROR	Bit 17 1 - Input Buffer descriptor pointer passed to codec is NULL when generateHeader is XDM_ENCODE_AU 0 - Error not found
	JPEG_NULL_INPUT_BUFFER_POINTER_ERROR	Bit 18 1 - Input Buffer pointer passed to codec is NULL when generateHeader is XDM_ENCODE_AU 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	JPEG_INVALID_INPUT_BUFFER_SIZE_ERROR	Bit 19 1 - Input buffer size is zero 0 - Error not found
	JPEG_INVALID_NUM_OF_INPUT_BUFFERS_ERROR	Bit 20 1 - Invalid number of input buffers (less than 1 OR greater than 3) passed to the codec when generateHeader is XDM_ENCODE_AU 0 - Error not found
	JPEG_INVALID_INPUT_BUFFER_MEMTYPE_ERROR	Bit 21 1 - Invalid input buffer memory type is passed to the codec when generateHeader is XDM_ENCODE_AU 0 - Error not found
	JPEG_INVALID_OUTPUT_BUFFER_MEMTYPE_ERROR	Bit 22 1 - Invalid output buffer memory type is passed to the codec 0 - Error not found
	JPEG_NULL_OUTARGS_POINTER_ERROR	Bit 23 1 - OutArgs Pointer passed to codec in process call is NULL 0 - Error not found
	JPEG_INVALID_OUTARGS_SIZE	Bit 24 1 - Invalid size of OutArgs passed to the codec 0 - Error not found
	JPEG_NULL_OUTPUT_BUF_DESC_ERROR	Bit 25 1 - Output Buffer descriptor pointer passed to codec is NULL 0 - Error not found
	JPEG_NULL_OUTPUT_BUFFER_POINTER_ERROR	Bit 26 1 - Output Buffer pointer passed to codec is NULL 0 - Error not found
	JPEG_INVALID_OUTPUT_BUFFER_SIZE_ERROR	Bit 27 1 - Output buffer size is zero 0 - Error not found
	JPEG_INVALID_NUM_OF_OUTPUT_BUFFERS_ERROR	Bit 28 1 - Number of output buffers passed to the codec is not 1 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	JPEG_INSUFFICIENT_OUTPUT_BUFFER_SIZE_ERROR	Bit 29 1 - Number of bytes encoded is greater than the number of bytes allocated to output buffer 0 - Error not found
	JPEG_INVALID_JFIF_THUMBNAIL_ENABLE_ERROR	Bit 30 1 - Invalid thumbnailIndexApp0 value (neither 0 nor 1) passed to codec through InArgs in process call 0 - Error not found
	JPEG_INVALID_EXIF_THUMBNAIL_ENABLE_ERROR	Bit 31 1 - Invalid thumbnailIndexApp1 value (neither 0 nor 1) passed to codec through InArgs in process call 0 - Error not found
	JPEG_INPUT_BUFFER_POINTER_ALIGN_ERROR	Bit 32 1 - The base address of the input 2D buffer in TILER region is not aligned to 16 bytes 0 - Error not found
	JPEG_DATASYNC_GET_ROW_DATA_ERROR	Bit 33 1 - numBlocks was set to a value less than 1 by input data sync call back function (getDataFxn) 0 - Error not found
	JPEG_DATASYNC_INVALID_RESTART_INTERVAL_ERROR	Bit 34 1 - Invalid Restart Interval when Data Sync (Slice Mode) is enabled 0 - Error not found
	JPEG_DATASYNC_BLOCK_POINTER_ERROR	Bit 35 1 - Invalid Buffer Pointer in the output data sync (getBufferFxn) call back function 0 - Error not found
	JPEG_DATASYNC_BLOCK_SIZE_ERROR	Bit 36 1 - Invalid Buffer Size in the output data sync (getBufferFxn) call back function 0 - Error not found
	JPEG_DATASYNC_INVALID_BLOCKS_ERROR	Bit 37 1 - Invalid Buffer Count in the output data sync (getBufferFxn) call back function 0 - Error not found

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	JPEG_DATASYNC_NOT_VALID_COMBINATION_ERROR	Bit 38 1 - Invalid Combination of scatteredBlocksFlag and varBlockSizesFlag in the output data sync (getBufferFxn) call back function 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 Encoder.

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
- IVIDENC2_Fxns
- IVIDENC2_Params
- IVIDENC2_DynamicParams
- IVIDENC2_InArgs
- IVIDENC2_Status
- IVIDENC2_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	XDM2_BufSize	Input	Size of the buffer in bytes

4.2.1.3 XDM2_BufDesc

‖ Description

This structure defines the buffer descriptor for output buffers.

‖ Fields

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

4.2.1.4 XDM1_AlgBufInfo

Description

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

Fields

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

Note:

For MJPEG Encoder, the buffer details are:

- Number of input buffers required is based on input chroma format.
- Number of output buffers required is 1.
- For frame mode of operation, there is no restriction on input buffer size except that it should contain atleast one frame data.
- The memory types supported for the output buffer are `XDM_MEMTYPE_RAW` and `XDM_MEMTYPE_TILEDPAGE`.
- The memory types supported for luma input buffers are `XDM_MEMTYPE_TILED8`, `XDM_MEMTYPE_TILEDPAGE` and `XDM_MEMTYPE_RAW`.
- The memory types supported for chroma input 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 input side
  - scatteredBlocksFlag
  - baseAddr
  - varBlockSizesFlag
  - blockSizes
- There are two modes of operations in data sync on the input side
  - NUMROWS Mode (IVIDEO_NUMROWS)
  - Entire Frame Mode (IVIDEO_ENTIREFRAME) (without data sync)
- Only numBlocks (non-zero) is updated by the application. Remaining parameters are not updated (don't care).
- There are three modes of operation in data sync on the output side
  - Slice Mode (IVIDEO_SLICEMODE)
  - Fixed Length Mode (IVIDEO_FIXEDLENGTH)
  - Entire Frame Mode (IVIDEO_ENTIREFRAME) (without data sync)
- In Slice Mode the following parameters are updated/supported as follows.
  - scatteredBlockFlag should be FALSE (bitstream buffer is assumed to be continuous).
  - varBlockSizesFlag should be FALSE.
  - numBlocks can be any positive number between 1 to 8.

- restartInterval (present in IJPEGVENC_DynamicParams) should not be set to 0 if IVIDEO_SLICEMODE is used.

- In Fixed Length Mode the following parameters are updated/supported as follows.
  - scatteredBlockFlag may be TRUE/FALSE.
  - varBlockSizesFlag may be TRUE/FALSE.
  - numBlocks may be any value between 1 and 8.
  - During the first Data Sync the data provided need not to be multiple of Page Size (1024 bytes).
  - Total size per Data Sync call (except the first call) should be multiple of Page Size (1024 bytes).
- In Entire Frame Mode, XDM_DataSyncDesc structure is ignored by the codec.

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]	XDM1_SingleBufDesc	Input/ Output	Description for video planes
metadataPlaneDesc [IVIDEO_MAX_NUM_METADATA_PLANES]	XDM1_SingleBufDesc	Input/ Output	Description for metadata planes
secondFieldOffsetWidth[IVIDEO_MAX_NUM_PLANES]	XDAS_Int32	Input/ Output	Off set value for second field in planeDesc buffer (width in pixels)
secondFieldOffsetHeight[IVIDEO_MAX_NUM_PLANES]	XDAS_Int32	Input/ Output	Off set 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

Field	Data Type	Input/ Output	Description
activeFrameRegion	XDM_Rect	Input/ Output	Actual display region/capture region
extendedError	XDAS_Int32	Input/ Output	Provision for informing the error type if any
frameType	XDAS_Int32	Input/ Output	Video frame types. See enumeration IVIDEO_FrameType. Not applicable for MJPEG encoder.
topFieldFirstFlag	XDAS_Int32	Input/ Output	Indicates when the application (should display)/(had captured) the top field first. Not applicable for MJPEG encoder.
repeatFirstFieldFlag	XDAS_Int32	Input/ Output	Indicates when the first field should be repeated. Not applicable for MJPEG encoder.
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 encoder.
contentType	XDAS_Int32	Input/ Output	Video content type. See IVIDEO_ContentType.
chromaFormat	XDAS_Int32	Input/ Output	Chroma format for encoder input data/decoded output buffer. See XDM_ChromaFormat enumeration for details.
scalingWidth	XDAS_Int32	Input/ Output	Scaled image width for post processing. Not applicable for MJPEG encoder.
scalingHeight	XDAS_Int32	Input/ Output	Scaled image height for post processing. Not applicable for MJPEG encoder.
rangeMappingLuma	XDAS_Int32	Input/ Output	Not applicable for MJPEG encoder
rangeMappingChroma	XDAS_Int32	Input/ Output	Not applicable for MJPEG encoder
enableRangeReductionFlag	XDAS_Int32	Input/ Output	ON/OFF, default is OFF. Not applicable for MJPEG encoder.

Note:

- `IVIDEO_MAX_NUM_PLANES`: Max YUV buffers - one each for Y, U, and V.
- The following parameters are not supported in this version of the encoder
  - `frameType`
  - `topFieldFirstFlag`
  - `repeatFirstFieldFlag`
  - `repeatFrame`
  - `scalingWidth`
  - `scalingHeight`
  - `rangeMappingLuma`
  - `rangeMappingChroma`
  - `enableRangeReductionFlag`

4.2.1.7 IVIDENC2_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 IVIDENC2_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.
encodingPreset	XDAS_Int32	Input	Not supported in MJPEG encoder.
rateControlPreset	XDAS_Int32	Input	Not supported in MJPEG encoder.
maxHeight	XDAS_Int32	Input	Maximum video height in pixels. The supported range is [32, 4096]. The default value is 1088.
maxWidth	XDAS_Int32	Input	Maximum video width in pixels. The supported range is [32, 4320]. The default value is 1920.
dataEndianness	XDAS_Int32	Input	Endianness of output data. Not supported. The output is always of type XDM_BYTE.

Field	Data Type	Input/ Output	Description
maxInterFrameInterval	XDAS_Int32	Input	I to P frame distance. Not applicable for MJPEG encoder.
maxBitRate	XDAS_Int32	Input	Maximum Bit-rate for encoding in bits per second. Not applicable for MJPEG encoder.
minBitRate	XDAS_Int32	Input	Minimum Bit-rate for encoding in bits per second. Not applicable for MJPEG encoder.
inputChromaFormat	XDAS_Int32	Input	Chroma format for the input buffer. Supported values are XDM_YUV_422IBE, XDM_YUV_444P, XDM_GRAY, and XDM_YUV_420SP. Default value is XDM_YUV_420SP.
inputContentType	XDAS_Int32	Input	Video content type of the buffer being encoded. Not applicable for MJPEG encoder.
operatingMode	XDAS_Int32	Input	Video coding mode of operation. Only IVIDEO_ENCODE_ONLY is supported.
profile	XDAS_Int32	Input	Profile indicator of video codec. Not applicable for MJPEG encoder.
level	XDAS_Int32	Input	Level indicator of video codec. Not applicable for MJPEG encoder.
inputDataMode	XDAS_Int32	Input	Input data mode. For encoder, the supported values are IVIDEO_NUMROWS and IVIDEO_ENTIREFRAME. Default value is IVIDEO_ENTIREFRAME.
outputDataMode	XDAS_Int32	Input	Output data mode. For encoder, the supported values are IVIDEO_FIXEDLENGTH, IVIDEO_SLICEMODE and IVIDEO_ENTIREFRAME. Default value is IVIDEO_ENTIREFRAME.
numInputDataUnits	XDAS_Int32	Input	Number of input slices/rows. Not supported in this version of MJPEG encoder.
numOutputDataUnits	XDAS_Int32	Input	Number of output slices/rows. Units depend on the outputDataMode, like number of slices/rows/blocks etc. Ignored if outputDataMode is set to full frame mode. Default value is 1.
metadataType	XDAS_Int32[]	Input	Type of each metadata plane. Not supported in this version of MJPEG encoder.

Note:

- The maximum height and width supported is 4096 pixels.
- The minimum height and width supported is 32 pixels.
- `dataEndianness` field should be set to `XDM_BYTE`.

4.2.1.9 IVIDENC2_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.
`inputHeight`	`XDAS_Int32`	Input	Input frame height. The supported range is [32, 4096]. The default value is 1080.
`inputWidth`	`XDAS_Int32`	Input	Input frame width. The supported range is [32, 4096]. The default value is 1920.
`refFrameRate`	`XDAS_Int32`	Input	Reference, or input, frame rate in fps * 1000. For example, if ref frame rate is 30 frames per second, this field will be 30000. Not applicable for MJPEG encoder.
`targetFrameRate`	`XDAS_Int32`	Input	Target frame rate in fps * 1000. For example, if target frame rate is 30 frames per second, this field will be 30000. Not applicable for MJPEG encoder.
`targetBitRate`	`XDAS_Int32`	Input	Target bit rate in bits per second. Not applicable for MJPEG encoder.
`intraFrameInterval`	`XDAS_Int32`	Input	The number of frames between two I frames. Not applicable for MJPEG encoder.
`generateHeader`	`XDAS_Int32`	Input	Supported. Set it to `XDM_GENERATE_HEADER` to generate only the header. Set it to `XDM_ENCODE_AU` in other cases. The default value is `XDM_ENCODE_AU`.

Field	Data Type	Input/ Output	Description
captureWidth	XDAS_Int32	Input	DEFAULT(0): use imageWidth as pitch else use given capture width for pitch provided it is greater than image width. captureWidth should be greater than or equal to image width or should be set to 0. If the input buffer is in non-TILED region, this parameter should be multiple of 128 bytes.
forceFrame	XDAS_Int32	Input	Force the current (immediate) frame to be encoded as a specific frame type. Not applicable for MJPEG encoder.
interFrameInterval	XDAS_Int32	Input	Number of B frames between two reference frames; that is, the number of B frames between two P frames or I/P frames. Not applicable for MJPEG encoder.
mvAccuracy	XDAS_Int32	Input	Pixel Accuracy of the motion vector. Not applicable for MJPEG encoder.
sampleAspectRatioHeight	XDAS_Int32	Input	Not applicable for MJPEG encoder.
sampleAspectRatioWidth	XDAS_Int32	Input	Not applicable for MJPEG encoder.
ignoreOutbufSizeFlag	XDAS_Int32	Input	Not applicable for MJPEG encoder.
putDataFxn	XDM_DataSyncPutFxn	Input	DataSync call back function pointer for putData. This should not be NULL if output data sync (FIXEDLENGTH or SLICEMODE) is enabled. The default value is NULL.
putDataHandle	XDM_DataSyncHandle	Input	DataSync handle for putData. This parameter is not used by the encoder (“don’t care”).
getDataFxn	XDM_DataSyncGetFxn	Input	DataSync call back function pointer for getData. This should not be NULL if input data sync (NUMROWS) is enabled. The default value is NULL.
getDataHandle	XDM_DataSyncHandle	Input	DataSync handle for getData. This parameter is not used by the encoder (“don’t care”).
getBufferFxn	XDM_DataSyncGetBufferFxn	Input	DataSync call back function pointer for getBuffer. This should not be NULL if output data sync (FIXEDLENGTH or SLICEMODE) is enabled. The default value is NULL.
getBufferHandle	XDM_DataSyncHandle	Input	DataSync handle for getBuffer. This parameter is not used by the encoder (“don’t care”).

Field	Data Type	Input/ Output	Description
lateAcquireArg	XDAS_Int32	Input	Argument used during late acquire. The default value is IRES_HDVICP2_UNKNOWNLATEACQUIREARG.

4.2.1.10 IVIDENC2_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.
inputID	XDAS_Int32	Input	Identifier to attach with the corresponding input frames to be encoded. This is useful when frames require buffering (e.g. B frames), and to support buffer management. Zero (0) is not a supported inputID. This value is reserved for cases when there is no input buffer is provided.
control	XDAS_Int32	Input	Encoder control operations.

4.2.1.11 IVIDENC2_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.

Field	Data Type	Input/ Output	Description
data	XDM1_SingleBufDesc	Output	Buffer information structure for information passing buffer. Not Supported in this version of MJPEG encoder.
encodingPreset	XDAS_Int32	Output	Encoding preset. Not supported in MJPEG encoder.
rateControlPreset	XDAS_Int32	Output	Rate control preset. Not supported in MJPEG encoder.
maxInterFrameInterval	XDAS_Int32	Output	I to P frame distance. Not applicable for MJPEG encoder.
inputChromaFormat	XDAS_Int32	Output	Chroma format for the input buffer.
inputContentType	XDAS_Int32	Output	Video content type of the buffer being encoded. Not applicable for MJPEG encoder.
operatingMode	XDAS_Int32	Output	Video coding mode of operation.
profile	XDAS_Int32	Output	Profile indicator of video codec. Not applicable for MJPEG encoder.
level	XDAS_Int32	Output	Level indicator of video codec. Not applicable for MJPEG encoder.
inputDataMode	XDAS_Int32	Output	Input data mode. For encoder, it is row mode/entire frame.
outputDataMode	XDAS_Int32	Output	Output data mode. For encoder, it is fixed length/slice mode/entire frame.
numInputDataUnits	XDAS_Int32	Output	Number of input slices/rows. Not supported in this version of MJPEG encoder.
numOutputDataUnits	XDAS_Int32	Output	Number of output slices/rows. Units depend on the outputDataMode, like number of slices/rows/blocks etc. Ignored if outputDataMode is set to full frame mode.
configurationID	XDAS_Int32	Output	Configuration ID of given codec. This is based on the input stream & can be used by the framework to optimize the save/restore overhead of any resources used. This can be useful in multichannel use case scenarios.

Field	Data Type	Input/ Output	Description
bufInfo	XDM1_AlgBufInfo	Output	Input and output buffer information. This field provides the application with the algorithm's buffer requirements. The requirements may vary depending on the current configuration of the algorithm instance.
metadataType	XDAS_Int32[]	Input	Type of each metadata plane. Not supported in this version of MJPEG encoder.
encDynamicParams	IVIDENC2_DynamicParams	Output	Video encoder dynamic parameters.

4.2.1.12 IVIDENC2_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
bytesGenerated	XDAS_Int32	Output	Bytes generated per encode call
encodedFrameType	XDAS_Int32	Output	See enumeration IVIDEO_FrameType. Always IVIDEO_I_FRAME.
inputFrameSkip	XDAS_Int32	Output	See enumeration IVIDEO_SkipMode. Not applicable for MJPEG encoder.
freeBufID[IVIDEO2_MAX_IO_BUFFERS]	XDAS_Int32	Output	This is an array of inputID's corresponding to the buffers that have been unlocked in the current process call.
reconBufs	IVIDEO2_BufDesc	Output	Reconstruction frames. Not applicable for MJPEG encoder.

4.2.2 MJPEG Encoder Data Structures

This section includes the following MJPEG Encoder specific data structures:

- IJPEGVENC_Params
- IJPEGVENC_DynamicParams
- IJPEGVENC_CustomQuantTables
- IJPEGVENC_InArgs
- IJPEGVENC_Status
- IJPEGVENC_OutArgs

4.2.2.1 IJPEGVENC_Params

Description

This structure defines the creation parameters and any other implementation specific parameters for an MJPEG Encoder instance object. The creation parameters are defined in the XDM data structure, IVIDENC2_Params.

Fields

Field	Data Type	Input/ Output	Description
videnc2Params	IVIDENC2_Params	Input	See IVIDENC2_Params data structure for details.
maxThumbnailHSizeApp0	XDAS_UInt16	Input	Max Horizontal resolution for APP0 (JFIF) thumbnail. Not supported in this encoder. This does not support encoding of thumbnails. This encoder supports only *insertion* of encoded thumbnail data provided by the application.
maxThumbnailVSizeApp0	XDAS_UInt16	Input	Max Vertical resolution for APP0 (JFIF) thumbnail. Not supported in this encoder. This does not support encoding of thumbnails. This encoder supports only *insertion* of encoded thumbnail data provided by the application.
maxThumbnailHSizeApp1	XDAS_UInt16	Input	Max Horizontal resolution for APP1 (Exif) thumbnail. Not supported in this encoder. This does not support encoding of thumbnails. This encoder supports only *insertion* of encoded thumbnail data provided by the application.

Field	Data Type	Input/ Output	Description
maxThumbnailVSizeApp1	XDAS_UInt16	Input	Max Vertical resolution for APP1 (Exif) thumbnail. Not supported in this encoder. This does not support encoding of thumbnails. This encoder supports only *insertion* of encoded thumbnail data provided by the application.
debugTraceLevel	XDAS_UInt32	Input	Specifies the debug trace level. MJPEG Encoder supports till level 4. Each higher level logs more debug trace data.
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.

4.2.2.2 IJPEGVENC_DynamicParams

Description

This structure defines the run-time parameters and any other implementation specific parameters for an MJPEG encoder object. The run-time parameters are defined in the XDM data structure, IVIDENC2_DynamicParams.

Fields

Field	Data Type	Input/ Output	Description
videnc2DynamicParams	IVIDENC2_DynamicParams	Input	See IVIDENC2_DynamicParams data structure for details.
restartInterval	XDAS_Int32	Input	If a positive non-zero value is provided, the encoder inserts RST marker after the specified number of MCUs. If this parameter is not specified or a value of 0 is specified, no RST markers are inserted. The default value is 0.
qualityFactor	XDAS_Int32	Input	The quantization table is modified based on this parameter. Set this parameter to 2 for lowest quality and 97 for best quality. If this parameter is not specified, the encoder uses the example quantization table specified in the JPEG standard document. The default value is 50.

Field	Data Type	Input/ Output	Description
quantTable	Pointer to IJPEGVENC_CustomQuantTables	Input	Application should populate the custom quant table in a structure of type IJPEGVENC_CustomQuantTables and pass the pointer to it here. Set it to NULL if custom quant tables are not to be used. The default value is NULL.

Note:

The user should not use both the quality setting and custom quant tables features simultaneously. In case the user wants to use custom quant tables, the qualityFactor param must be set to 0.

4.2.2.3 IJPEGVENC_CustomQuantTables

‖ Description

This structure defines the custom quantization tables, if any, for the MJPEG Encoder.

‖ Fields

Field	Data Type	Input/ Output	Description
lumQuantTab	XDAS_UInt16[64]	Input	The array "lum_quant_tab" defines the quantization table for the luma component.
chmQuantTab	XDAS_UInt16[64]	Input	The array "chm_quant_tab" defines the quantization table for the chroma components.

4.2.2.4 IJPEGVENC_InArgs

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
videnc2InArgs	IVIDENC2_InArgs	Input	See IVIDENC2_InArgs data structure for details.

APPN0	XDM2_SingleBufDesc	Input	XDM2_SingleBufDesc Buffer containing JFIF thumbnail data. The application should provide only the encoded thumbnail data in this buffer. The JFIF header is appended by the encoder. The thumbnail data from this buffer is inserted as part of the JFIF Extension marker segment. The data in the buffer is copied “as is” to the bitstream after appending the APP0 Marker (0xFFE0), Segment Length and JFIF identifier code
thumbnailIndexApp0	XDAS_UInt16	Input	Set to 1 if APP0 (JFIF) marker needs to be inserted.
APPN1	XDM2_SingleBufDesc	Input	XDM2_SingleBufDesc Buffer containing Exif thumbnail data. The data in the buffer is copied “as is” to the bitstream after appending the APP1 Marker (0xFFE1), size of APP1 marker, Exif identifier code and TIFF header.
thumbnailIndexApp1	XDAS_UInt16	Input	Set to 1 if APP1 (Exif) marker needs to be inserted.
Comment	XDM2_SingleBufDesc	Input	XDM2_SingleBufDesc Buffer containing comment marker segment data. Set it to NULL if no comment needs to be inserted. The data in the buffer is copied “as is” to the bitstream after appending the Comment Marker start code (0xFFFE) and the size of the comment marker segment.

Note:

The maximum size supported for the JFIF, Exif and comment marker segments is 64 KB.

4.2.2.5 IJPEGVENC_Status

‖ Description

This structure defines parameters that describe the status of the MJPEG Encoder and any other implementation specific parameters. The `status` parameters are defined in the XDM data structure, `IVIDENC2_Status`.

‖ Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
videnc2Status	IVIDENC2_Status	Output	See IVIDENC2_Status data structure for details.
debugTraceLevel	XDAS_UInt32	Output	Specifies the debug trace level. MJPEG Encoder supports till level 4. Each higher level logs more debug trace data.
lastNFramesToLog	XDAS_UInt32	Output	Specifies the number of most recent frames to log in debug trace.
extMemoryDebugTraceAddr	XDAS_UInt32 *	Output	Specifies the address of the debug trace dump in external memory.
extMemoryDebugTraceSize	XDAS_UInt32	Output	Specifies the size of the debug trace dump in external memory.
extendedErrorCode0	XDAS_UInt32	Output	Parameter added to capture specific errors not captured in base Status structure
extendedErrorCode1	XDAS_UInt32	Output	Parameter added to capture specific errors not captured in base Status structure
extendedErrorCode2	XDAS_UInt32	Output	Parameter added to capture specific errors not captured in base Status structure
extendedErrorCode3	XDAS_UInt32	Output	Parameter added to capture specific errors not captured in base Status structure

4.2.2.6 IJPEGVENC_OutArgs

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
videnc2OutArgs	IVIDENC2_OutArgs	Output	See IVIDENC2_OutArgs data structure for details.

4.3 Interface Functions

This section describes the application programming interfaces used in the MJPEG Encoder. The MJPEG Encoder 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 Encoder. The initialization parameters are defined in the `IVIDENC2_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 Encoder during run-time. This is done by changing the status of the controllable parameters of the encoder during run-time. These controllable parameters are defined in the `IVIDENC2_DynamicParams` data structure (see Data Structures section for details).

‖ Name

`control()` – change run-time parameters of the MJPEG Encoder and query the encoder status

‖ Synopsis

```
XDAS_Int32 (*control)(IVIDENC2_Handle handle, IVIDENC2_Cmd
id,IVIDENC2_DynamicParams *params, IVIDENC2_Status
*status);
```

‖ Arguments

```
IVIDENC2_Handle handle; /* handle to the MJPEG encoder
instance */

IVIDENC2_Cmd id; /* MJPEG encoder specific control
commands*/

IVIDENC2_DynamicParams *params /* MJPEG encoder run-time
parameters */

IVIDENC2_Status *status /* MJPEG encoder 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 Encoder and queries the status of encoder. `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 Encoder instance object.

The second argument is a command ID. See `IVIDENC2_Cmd` in enumeration table for details.

The third and fourth arguments are pointers to the `IVIDENC2_DynamicParams` and `IVIDENC2_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 Encoder.

‖ 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 encoding call

‖ Synopsis

```
XDAS_Int32 (*process)(IVIDENC2_Handle handle,
IVIDEO2_BufDesc *inBufs, XDM2_BufDesc *outBufs,
IVIDENC2_InArgs *inargs, IVIDENC2_OutArgs *outargs);
```

‖ Arguments

```
IVIDENC2_Handle handle; /* handle to the MJPEG encoder
instance */

IVIDEO2_BufDesc *inBufs; /* pointer to input buffer
descriptor data structure */

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

IVIDENC2_InArgs *inargs /* pointer to the MJPEG encoder
runtime input arguments data structure */

IVIDENC2_OutArgs *outargs /* pointer to the MJPEG encoder
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 encoding. The first argument to `process()` is a handle to the MJPEG Encoder instance object.

The second and third arguments are pointers to the input and output buffer descriptor data structures respectively (see `IVIDEO2_BufDesc` and `XDM2_BufDesc` data structure for details).

The fourth argument is a pointer to the `IVIDENC2_InArgs` data structure that defines the run-time input arguments for the MJPEG Encoder instance object.

The last argument is a pointer to the `IVIDENC2_OutArgs` data structure that defines the run-time output arguments for the MJPEG Encoder 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 `IVIDENC2_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. 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 Encoder and free up the memory space that it uses.

‖ Name

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

‖ Synopsis

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

‖ Arguments

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

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

‖ Return Value

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

‖ Description

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

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

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

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

‖ See Also

`algAlloc()`

Chapter 5

Frequenty Asked Questions

This section answers frequently asked questions related to using MJPEG Encoder 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 IVIDENC2 interface.
Does MJPEG Encoder support non-multiple of 16 frame dimensions?	Yes, this encoder supports non-multiple of 16 image dimensions. Even odd resolutions are supported in this version.

Question	**Answer**
Does this MJPEG Encoder support custom quantization tables?	Yes.
Can I control image quality levels?	Yes. There are two ways of doing this: a. By specifying the quality factor for encoding, or b. Specifying a custom quantization table.
Does this MJPEG Encoder support custom Huffman tables?	No. This encoder always uses the Huffman tables suggested in Annex K of the JPEG Standard document.
Does Algorithm support DataSync mechanism for low-delay applications?	Yes. It has the mechanism for both input and output buffers.
Does this encoder support “generate header only” feature?	Yes.
Does this encoder support insertion of APPx markers?	Yes. The encoder supports insertion of JFIF, Exif, and Comment markers into the JPEG bitstream.
Does this encoder support insertion of restart (RST) markers?	Yes.
What are the maximum and minimum resolutions supported by the encoder?	This encoder supports resolutions ranging from 32x32 to 4096x4096.
What are the chroma formats supported for input?	The encoder supports YUV420 (Semi-Planar), YUV422 (YUYV), YUV444 (Planar) and YUV400 chroma formats for input.
Does the encoder support encoding in multiple scans?	No. The output of the encoder always contains a single scan.

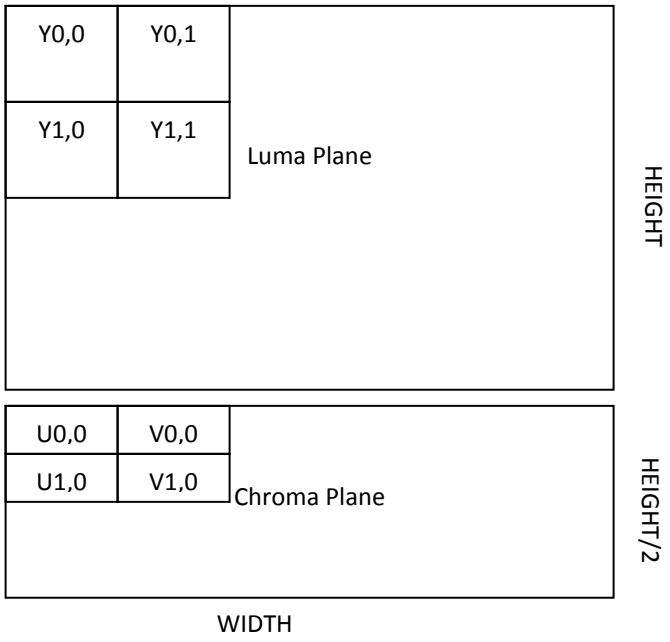
Chapter 6

Picture Format

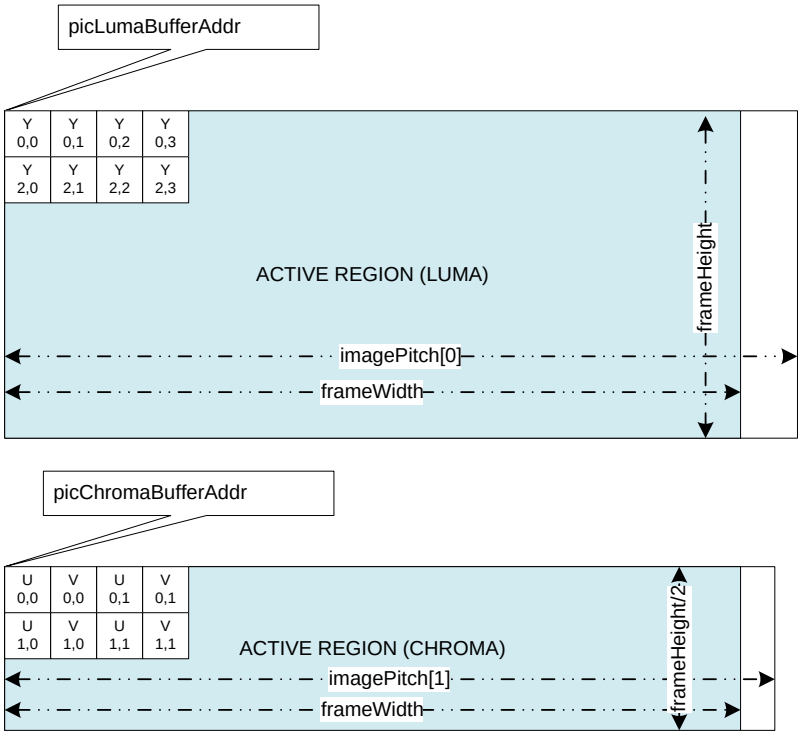
This chapter explains the picture format details for MJPEG Encoder. MJPEG Encoder expects the input uncompressed picture to be in one of the following formats: YUV444 Planar, YUV420 Semi-Planar (NV12), YUV422 YUYV or YUV400 format.

6.1 NV12 Chroma Format

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



6.2 Progressive Picture Format



ActiveRegion: Data to be encoded

Extra region beyond the ActiveRegion may be allocated by application due to imagePitch constraints.

Both luma and chroma buffers can be allocated independently and both can have their pitch different.

6.3 Constraints on Parameters

imagePitch need to comply with following constraints

- imagePitch shall be greater or equal to the Width (passed by the application host).
- imagePitch is “don’t care” if the buffer is in TILED8, TILED16 or TILED32 region

Buffer Addresses need to comply with following constraints

- addresses shown as picLumaBufferAddr in figures shouldn’t point to any region which is not TILED8 or RAW/TILED PAGE
- The addresses shown as picChromaBufferAddr in figures shouldn’t point to any region which is not TILED8, TILED16 or RAW/TILED PAGE

Constraints on resolutions are defined as below.

Minimum frameWidth = 32

Minimum frameHeight = 32

Maximum frameWidth = 4096

Maximum frameHeight = 4096

This encoder does not support interlaced input.

This page is intentionally left blank

Chapter 7

Debug Trace Usage

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

7.1 Introduction

This section explains This section explains the approach and overall design that will be adopted forenabling 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 MJPEG encoder create time parameters are modified as

```
typedef struct IJPEGVENC_Params {
  IVIDENC2_Params videnc2Params;
  XDAS_UInt16     maxThumbnailHSizeApp0;
  XDAS_UInt16     maxThumbnailVSizeApp0;
  XDAS_UInt16     maxThumbnailHSizeApp1;
  XDAS_UInt16     maxThumbnailVSizeApp1;
  XDAS_UInt32   debugTraceLevel;
```

```
  XDAS_UInt32   lastNFramesToLog;

} IJPEGVENC_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 IJPEGVENC_Status {

  IVIDENC2_Status videnc2Status;

  XDAS_UInt32 debugTraceLevel;

  XDAS_UInt32 lastNFramesToLog;

  XDAS_UInt32 * extMemoryDebugTraceAddr;

  XDAS_UInt32 extMemoryDebugTraceSize;

  XDAS_UInt32 extendedErrorCode0;

  XDAS_UInt32 extendedErrorCode1;

  XDAS_UInt32 extendedErrorCode2;

  XDAS_UInt32 extendedErrorCode3;

} IJPEGVENC_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 encoder 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 IVIDENC2 interface.

8.2 MJPEG Encoder Input with Sub-frame Level Synchronization

This section explains the IVIDENC2 interface details, which help to achieve the sub-frame level communications.

Table 7-1, Table 7-2 and Table 7-3 explain the creation, control and handshake parameters related to sub frame level data communication for input data of MJPEG Encoder respectively.

"Details" column is a generic column and "valid values" column is specific to video encoder input.

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

Parameter Name	Details	Valid Values
`IVIDENC2_Params::inputDataMode`	Defines the mode of accepting the input frame.	❑ `IVIDEO_ENTIREFRAME`: Entire frame data is given to encoder ❑ `IVIDEO_NUMROWS`: Frame data is given in unit of Number of MB rows, each MB row is 16 lines of video data.
`IVIDENC2_Params::numInputDataUnits`	Unit of input data	Don't care. As the information about the data can be available during sub frame level communication.

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

Parameter Name	Details	Valid Values
IVIDENC2_DynamicParams::getDataFxn	This function pointer is provided by the app/framework to the MJPEG Encoder. The encoder calls this function to get partial video buffer(s) from the app/framework. Apps/frameworks that support datasync should set this to non-NULL.	Any non-NULL value if inputDataMode != IVIDEO_ENTIREFRAME
IVIDENC2_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 VIDENC2_DynamicParams.getDataFxn() The app/framework can use this handle to differentiate callbacks form different algorithms.	Any Value

Table 8-3 Handshake Parameters Related to Sub-frame Level Data Communication for Input Data of MJPEG Encoder

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 as buffer is assumed to be contiguous.
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 since it is assumed to be contiguous yuv buffer and initial address is available at process call.
XDM_DataSyncDesc::numBlocks	Number of data blocks	Any positive value. If less than 1, then it is an error.
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 the application wants to use MJPEG Encoder to operate with sub frame on input side:

- It should create the MJPEG Encoder with IVIDENC2_Params::inputDataMode = IVIDEO_NUMROWS.
- It should also make a control call with `IVIDENC2_DynamicParams::getDataFxn` = non-NULL; to use sub frame level data communication, control call is mandatory.
- It should provide the base address of the input buffer during process call.
- It should provide all the data availability via getDataFxn call back, during process call the input buffer is assumed to be data-less.
- `IVIDENC2_DynamicParams::getDataFxn == NULL` && `IVIDENC2_Params::inputDataMode == IVIDEO_NUMROWS` is an erroneous situation and codec returns error during process call.

8.3 MJPEG Encoder Output with Sub-frame Level Synchronization

This section explains the IVIDENC2 interface details, which help to achieve the sub frame level data synchronization for output.

Table 7-4, Table 7-5 and Table 7-6 explain the creation and control parameters related to sub frame level data communication for output data of MJPEG Encoder respectively.

“Details” column is a generic column and “valid values” column is specific to MJPEG Encoder output.

Table 8-4 Creation Time Parameter Related to Sub-frame Level Data Communication for Output Data of MJPEG Encoder

Parameter Name	Details	Valid Values
`IVIDENC2_Params::outputDataMode`	Defines the mode of providing the output data.	`IVIDEO_ENTIREFRAME` : Entire frame bitstream is given out by the encoder `IVIDEO_FIXEDLENGTH`: bit-stream is provided by encoder after a fixed length of bytes. The length has to be multiple of 1K. `IVIDEO_SLICEMODE`: bit-stream is provided by encoder after producing a single(or more) number of slice units
`IVIDENC2_Params::numOutputDataUnits`	Unit of output data	Don’t care if `inputDataMode == IVIDEO_ENTIREFRAME` Any positive value if `outputDataMode != IVIDEO_ENTIREFRAME`. If `outputDataMode ==`

		IVIDEO_FIXEDLENGTH then it indicates the basic unit of size (in multiple of 1K) at which encoder should inform the application. For example: Here 4 means that encoder should inform after producing every 4*1024 bytes to application if outputDataMode == IVIDEO_SLICEMODE then it indicates the basic unit of slices at which encoder should produce the bit- stream. For example: Here 5 means that after encoding a set of 5 slices, encoder should inform to application

Table 8-5 Dynamic parameters related to sub frame level data communication for output data of MJPEG encoder

Parameter Name	Details	Valid Values
IVIDENC2_DynamicParams::putDataFxn	This function pointer is provided by the app/framework to the MJPEG Encoder. The encoder calls this function when data has been put in output buffer. It is to inform the app/framework. Apps/frameworks that support datasync should set this to non-NULL.	Any non-NULL value if outputDataMode != IVIDEO_ENTIREFRAME
IVIDENC2_DynamicParams::putDataHandle	It defines the handle to be used while informing data availability to application. This is a handle which codec must provide when calling putDataFxn. Apps/frameworks that support datasync should set this to non-NULL. For an algorithm, this handle is read-only; it must not be modified when calling the appregistered. IVIDENC2_DynamicParams.putDataFxn(). The app/framework can use this handle to differentiate callbacks from different algorithms.	Any Value.

To simplify the codec implementation, the information sharing by codec to application happens at a quantum of 1 Kbyte data. In this document, each 1 Kbyte is referred as page.

If application wants to use MJPEG Encoder to operate with sub-frame data sync on output side:

- It should create the video encoder with IVIDENC2_Params::outputDataMode = IVIDEO_SLICEMODE or IVIDEO_FIXEDLENGTH.

- It should also make a control call with `IVIDENC2_DynamicParams::putDataFxn` = non-NULL; to use sub frame level data communication, control call is mandatory.
- It should provide the base address and available space of the output buffer during process call.
- `IVIDENC2_DynamicParams::putDataFxn` == NULL && `IVIDENC2_Params::outputDataMode` != `IVIDEO_ENTIREFRAME` is an erroneous situation and codec returns error during process call.

8.3.1 For outputDataMode Equal to IVIDEO_SLICEMODE

Table 7-6 explains the handshake parameters related to sub frame level data communication (*IVIDEO_SLICE* mode) for output data of MJPEG Encoder respectively. In case of outputDataMode = IVIDEO_SLICEMODE, following points should be noted.

- When *IVIDEO_SLICE* mode is enabled, ensure that the Restart Interval is a positive value. Enabling *IVIDEO_SLICE* mode without Restart Interval is an error.
- Communication point by codec to application happens when number of slices equals to numOutputDataUnit. i.e. if `numOfSlice == numOutputDataUnit` then make a putData call.
- `numOutputDataUnit` is the frequency after which codec will inform to application. If 'outputDataUnit' is 8 then, after 8 slice codec has to make putData call. Larger the number of `outputDataUnit`, larger the size requirement of encoder in SL2 to retain the information for each slice. So to keep the SL2 size impact minimal, TI's encoder implementations has constraint of limiting maximum allowed value of `outputDataUnit` as 8.
- bit-stream is assumed to be contiguous in memory, hence the bit-stream address is obtained during process call and the `XDM_DataSyncDesc::baseAddr` is don't care. It is a constraint of TI's encoder implementation.
- Application provides buffer size and address for bit-stream during process call, both of them are honored and consumed by encoder until it needs more space to write bit-stream (refer getBuf interface of MJPEG Encoder for more details)
- All data availability is informed via data synch calls, while process exit the bytesGenerated indicates the total sum (not the size of last chunk).

Table 8-6 Handshake parameters related to sub frame level data communication for output data of MJPEG encoder (outputDataMode = 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.	XDAS_FALSE. ***Constraint**: This will be XDAS_FALSE as bit stream memory is assumed to be continuous in case of IVIDEO_SLICEMODE mode.*
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***
XDM_DataSyncDesc::numBlocks	Number of data blocks	Should be always equal to '1'
XDM_DataSyncDesc::varBlockSizeFlag	Flag indicating whether any of the data blocks vary in size.	XDAS_FALSE
XDM_DataSyncDesc::blockSizes	Variable block sizes array.	If varBlockSizesFlag is XDAS_TRUE, this array contains the sizes of each slice. So total slice size is sum of (blockSizes[0] to blockSizes[numBlocks -1]. If varBlockSizesFlag is XDAS_FALSE, this contains the size of same-size slices. So total data given by encoder to app would be (numBlocks * blocSizes[0])

8.3.2 For outputDataMode Equal to IVIDEO_FIXEDLENGTH

Table 7-7 explains the handshake parameters related to sub frame level data communication (*IVIDEO_FIXEDLENGTH* mode) for output data of MJPEG Encoder respectively. In case of outputDataMode = IVIDEO_FIXEDLENGTH, following points should be noted.

- Communication point is one of the below whichever is **earlier.**
  - if 8 non-continuous blocks have been generated by encoder.
  - 1Kb * numOutputDataUnit of data is encoded.

- numOututDataUnit is the frequency after which codec will inform to App. so in IVIDEO_FIXED_LENGTH, lets outputDataUnit is 10 then after 10 page cross over (which is communication point to app) in SL2 bitstream space, codec will make putData call. if numOutputDataUnit is 10, and initial bitstream buffer size given in process call is 0.5 KB, then codec will put a putData call after 9.5 kB of encoding, not after 10.5 kB.

- Application provides buffer size and address for bit-stream during process call, both of them are honored and consumed by encoder until it needs more space to write bit-stream (refer getBuf interface of MJPEG Encoder for more details).

- All data availability is informed via data synch calls, while process exit the bytesGenerated indicates the total sum (not the size of last chunk).

Table 8-7 Handshake parameters related to sub frame level data communication for output data of MJPEG encoder (outputDataMode = 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.	Flag indicating whether the individual data block 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.	Base address of single data block or pointer to an array of 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 i.e. from baseAddr[0] to baseAddr[numBlocks-1].
XDM_DataSyncDesc::numBlocks	Number of data blocks	It is the number of blocks generated till the point of putData call. 1 <= numBlocks <= 8
XDM_DataSyncDesc::varBlockSizeFlag	Flag indicating whether any of the data blocks vary in size.	Flag indicating whether any of the data blocks vary in size. Valid values XDAS_TRUE or XDAS_FALSE.

XDM_DataSyncDesc::blockSizes	Variable block sizes array.	If varBlockSizesFlag is XDAS_TRUE, this array contains the sizes of each block. So total data size or bitstream is sum of (blockSizes[0] to blockSizes[numBlocks -1]. If varBlockSizesFlag is XDAS_FALSE, this contains the size of same-size data blocks.So total data given by encoder to app would be (numBlocks * blocSizes[0]).

8.4 MJPEG Encoder with partial buffer on output side

With IVIDENC2 interface, MJPEG Encoder can work with a situation when it has not been provided complete bit-stream buffer to it during process call. Application can provide non contiguous chunks of memory with some size constraints to encoder and it can produce the bit-stream in these buffers. It is achieved by IVIDENC2_DynamicParams::getBufFxn() interface. To get the encoder working with partial output buffer, there is no specific creation time parameter. Control call is mandatory and application need to provide a valid function pointer as IVIDENC2_DynamicParams::getBufFxn.

Table 7-8 and Table 7-9 explains the control and handshake parameters related to sub frame level data communication to handle partial output buffer by MJPEG Encoder respectively.

“Details” column is a generic column and “valid values” column is specific to MJPEG Encoder.

Table 8-8 Dynamic parameters related to accept partial buffer for output bit-stream

Parameter Name	Details	Valid values
IVIDENC2_DynamicParams::getBufFxn	This function pointer is provided by the app/framework to the MJPEG Encoder. The encoder calls this function to get partial bit-stream buffer(s) from the app/framework. Apps/frameworks that support datasync should set this to non-NULL.	Any non-NULL value to use partial buffer for bit-stream space
IVIDENC2_DynamicParams::getDataHandle	This is a handle which the codec must provide when calling the app-registered IVIDENC2_DynamicParam.getBufferFxn(). Apps/frameworks that don't support datasync should set this to NULL. For an algorithm, this handle is read-only; it must not be modified when calling the app-registered IVIDENC2_DynamicParams.getBufferFxn(). The app/framework can use this handle to differentiate callbacks from different algorithms.	Any Value

Table 8-9 Handshake parameters related to accept partial buffer for output bit-stream

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.	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.	non-NULL, in case if IVIDEO_FIXED_LENGTH mode. Don't care in IVIDEO_SLICE mode.
XDM_DataSyncDesc::numBlocks	Number of data blocks	1<= numBlocks <= 8.
XDM_DataSyncDesc::varBlockSizeFlag	Flag indicating whether any of the data blocks vary in size.	XDAS_TRUE or XDAS_FALSE
XDM_DataSyncDesc::blockSizes	Variable block sizes array.	blockSizes[i] should be always multiple of 1K. totalBlockSize = SUM (blockSizes[0] to blockSizes[numBlocks-1]) if varBlockSizesFlag is non zero. totalBlockSize = numBlocks * blockSizes[0] if varBlockSizesFlag is zero.

Following points should be noticed to use MJPEG Encoder with partial buffer on output side:

- getBuf is independent of outputDataMode or inputDataMode. It is only meant for codec to ask application for a buffer, if encoder has exhausted for output bit-stream.
- During process call the initial stream address and size are provided by application. The following are some constraints in providing the initial stream buffer.

- In FIXEDLENGTH mode, the initial stream buffer size should not be less than the page size (1 KB).
- In SLICEMODE, the initial stream buffer size should be multiple of page size (1 KB).

- During data synch (via getBuf) codec can accept a multiple non contiguous buffers from application each of them has to be multiple of 1K.

- In FIXEDLENGTH mode, when Thumbnail or Comment Markers are enabled, the initial stream buffer size should be greater than or equal to the sum of twice the page size, Thumbnail and Comment sizes. For example, if the thumbnail size is 4.2 KB and the comment marker size is 1.4 KB, then the initial stream buffer size should be greater than or equal to 7.6 KB (1 * 2 + 4.2 + 1.4 = 7.6 KB).

- In SLICEMODE, when Thumbnail or Comment Markers are enabled, the initial stream buffer size should be greater than or equal to the sum of twice the page size, thumbnail and comment sizes. In addition, the initial stream buffer size should be multiple of page size (1 KB).

- In FIXEDLENGTH mode, if scatteredBlocksFlag is FALSE and varBlockSizeFlag is TRUE, then this combination is treated as an ERROR case.

- if scatteredBlocksFlag is non zero

  ***Constraint**: Maximum number of blocks provided by user should be 8. If application provides more than 8 block then codec will just accept 8 blocks and rest of the blocks will be ignored.*

If the function pointer `IVIDENC2_DynamicParams::getBufFxn` provided is null then encoder will first consume the buffer provided in process call (by writing the bit stream data), if that buffer is exhausted then encoder has to do proper pipe down and come out from the process call with error.

Chapter 9

Error Handling

This section explains the error handling by MJPEG encoder.

9.1 Description

This version of the encoder supports handling of erroneous situations while encoding. If encoder encounters any erroneous situations, it shall exit gracefully without any hang or crash. Also, encoder process call shall return `IVIDENC2_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 encoder. 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 IVIDENC2_OutArgs and IVIDENC2_Status

Bit	Error Code	Explanation	Recommended App Behaviour
0	`IJPEGVENC_ERR_UNSUPPORTED_VIDENC2PARAMS`	This error code has been deprecated	NA
1	`IJPEGVENC_ERR_UNSUPPORTED_VIDENC2DYNAMICPARAMS`	Unsupported VIDENC2DynamicParams are passed to the codec	Call GETSTATUS control by passing extended Status structure to get more details about the error through `extendedErrorCode0` or `extendedErrorCode1` parameters. Then, refer Table 8-2 for recommended app behaviour.
2	`IJPEGVENC_ERR_UNSUPPORTED_JPEGENCDYNAMICPARAMS`	Unsupported JPEGVENC_DynamicParams are passed to the codec	Call GETSTATUS control by passing extended Status structure to get more details about the error through `extendedErrorCode0` or `extendedErrorCode1` parameters. Then, refer Table 8-2 for recommended app behaviour.
3	`IJPEGVENC_ERR_IMPROPER_DATASYNC_SETTING`	This error code has been deprecated	NA
4	`IJPEGVENC_ERR_NOSLICE`	This error code has been deprecated	NA

5	IJPEGVENC_ERR_SLICEHDR	This error code has been deprecated	NA
6	IJPEGVENC_ERR_MBDATA	This error code has been deprecated	NA
7	IJPEGVENC_ERR_UNSUPPFEATURE	This error code has been deprecated	NA
16	IJPEGVENC_ERR_STREAM_END	This error code has been deprecated	NA
17	IJPEGVENC_ERR_INVALID_MBOX_MESSAGE	Invalid MailBox Message has been received	This error will occur only when number of mail box messages from HDVICP2 to Media Controller exceeds (2^{32} - 1).
18	IJPEGVENC_ERR_HDVICP_RESET	This error code has been deprecated	NA
19	IJPEGVENC_ERR_HDVICP_WAIT_NOT_CLEAN_EXIT	Exit from HDVICP2 is not clean	
20	IJPEGVENC_ERR_IRES_RESHANDLE	This error code has been deprecated	NA
21	IJPEGVENC_ERR_STANDBY	HDVICP was not in standby when given to codec	Put HDVICP2 in standby and invoke process call again.
22	IJPEGVENC_ERR_INPUT_DATASYNC	Error in the Input Data Sync Call Function	Call GETSTATUS control by passing extended Status structure to get more details about the error through extendedErrorCode0 or extendedErrorCode1 parameters. Then, refer Table 8-2 for recommended app behaviour.
23	IJPEGVENC_ERR_OUTPUT_DATASYNC	Error in the Output Data Sync Call Function	Call GETSTATUS control by passing extended Status structure to get more details about the error through extendedErrorCode0 or extendedErrorCode1 parameters. Then, refer Table 8-2 for recommended app behaviour.

Table 9-2 Error Codes used to set the extendedErrorCode0 and extendedErrorCode1 fields in IJPEGVENC_Status

Bit	Error Code	Explanation	XDM Error Code Mapping	Recommended App Behaviour
0	JPEG_DYNAMIC_PARAMS_HANDLE_ERROR	Dynamic Params pointer passed to codec is NULL	XDM_FATALERROR	Invoke control call again with proper DynamicParams structure
1	JPEG_STATUS_HANDLE_ERROR	Status Pointer passed to codec is NULL	XDM_FATALERROR	Invoke control call again with proper Status structure
2	JPEG_DYNAMIC_PARAMS_SIZE_ERROR	Invalid size of dynamic parameters passed to codec	XDM_FATALERROR	Invoke control call again with proper DynamicParams structure
3	JPEG_ENCODE_HEADER_ERROR	Invalid GenerateHeader value passed to the codec	No XDM mapping	Invoke control call again with proper value for generateHeader
4	JPEG_UNSUPPORTED_RESOLUTION	Frame height and Frame width passed to the codec is less than 32 or greater than Max Width and Max Height provided during create time	No XDM Mapping	Invoke control call again with proper values for frame width and frame height
5	JPEG_CAPTURE_WIDTH_ERROR	Invalid Capture Width value passed to the codec	No XDM Mapping	Invoke control call again with proper value for captureWidth
6	JPEG_GET_DATA_FXN_NULL_POINTER	No call back function pointer is passed for getDataFxn	No XDM Mapping	Invoke control call again with proper pointer for getDataFxn
7	JPEG_GET_BUFFER_FXN_NULL_POINTER	No call back function pointer is passed for getBufferFxn OR putDataFxn	No XDM Mapping	Invoke control call again with proper pointer for getBufferFxn and putDataFxn

8	JPEG_INVALID_RESTART_INTERVAL_ERROR	Invalid Restart Interval value (< 0) passed to the codec	No XDM Mapping	Codec does not insert any restart marker in this case. Invoke control call again with proper value for restart interval if you want restart marker to be inserted.
9	JPEG_INVALID_QUALITY_FACTOR_ERROR	Invalid Quality factor value passed to the codec. Valid range is [1, 100].	No XDM Mapping	Codec uses quality factor of 50 by default. Invoke control call again with proper value for quality factor if you want to encode with another quality factor.
10	JPEG_INVALID_INPUT_CHROMA_FORMAT_ERROR	Invalid chroma format passed to the codec	No XDM Mapping	Invoke control call again with proper value for input chroma format.
11	JPEG_NULL_QUANT_TABLE_POINTER_ERROR	Both quality factor and user defined quantization table are valid	No XDM Mapping	Invoke control call again with either proper quality factor or user defined quantization table but not both.
12	JPEG_NULL_INARGS_POINTER_ERROR	InArgs Pointer passed to codec in process call is NULL	XDM_FATALERROR	Invoke process call again with proper IJPEGVENC_InArgs pointer.
13	JPEG_NULL_INARGS_APP_POINTER_ERROR	Both APP0 & APP1 Segments are passed to the codec	XDM_FATALERROR	Invoke process call again either with APP0 pointer or APP1 pointer but not both.
14	JPEG_INARGS_SIZE_ERROR	Invalid size of InArgs passed to the codec	XDM_FATALERROR	Invoke process call again with proper InArgs size
16	JPEG_INVALID_INPUT_ID_ERROR	Value of 0 was passed as input ID	XDM_FATALERROR	Invoke process call again with proper input ID if you want to encode a frame
17	JPEG_NULL_INPUT_BUF_DESC_ERROR	Input Buffer descriptor pointer passed to codec is NULL when generateHeader is XDM_ENCODE_AU	XDM_FATALERROR	Invoke process call again with proper IVIDEO2_BufDesc structure pointer
18	JPEG_NULL_INPUT_BUFFER_POINTER_ERROR	Input Buffer pointer passed to codec is NULL when generateHeader is XDM_ENCODE_AU	XDM_FATALERROR	Invoke process call again with input plane descriptor buffer pointer (planeDesc.buf) in the IVIDEO2_BufDesc structure.
19	JPEG_INVALID_INPUT_BUFFER_SIZE_ERROR	Input buffer size is zero	XDM_FATALERROR	Invoke process call again with proper input buffer sizes.
20	JPEG_INVALID_NUM_OF_INPUT_BUFFERS_ERROR	Invalid number of input buffers (less than 1 OR greater than 3) passed to the codec when generateHeader is XDM_ENCODE_AU	XDM_FATALERROR	Invoke process call again with proper number of input planes.
21	JPEG_INVALID_INPUT_BUFFER_MEMTYPE_ERROR	Invalid input buffer memory type is passed to the codec when generateHeader is XDM_ENCODE_AU	XDM_FATALERROR	Invoke process call again with proper memory type of the input buffer.
22	JPEG_INVALID_OUTPUT_BUFFER_MEMTYPE_ERROR	Invalid output buffer memory type is passed to the codec	XDM_FATALERROR	Invoke process call again with proper memory type of the output buffer.
23	JPEG_NULL_OUTARGS_POINTER_ERROR	OutArgs Pointer passed to codec in process call is NULL	XDM_FATALERROR	Invoke process call again with proper IVIDENC2_OutArgs pointer.
24	JPEG_INVALID_OUTARGS_SIZE	Invalid size of OutArgs passed to the codec	XDM_FATALERROR	Invoke process call again with proper OutArgs size
25	JPEG_NULL_OUTPUT_BUF_DESC_ERROR	Output Buffer descriptor pointer passed to codec is NULL	XDM_FATALERROR	Invoke process call again with output buffer descriptor pointer in the XDM2_BufDesc structure.

26	JPEG_NULL_OUTPUT_BUFFER_POINTER_ERROR	Output Buffer pointer passed to codec is NULL	XDM_FATALERROR	Invoke process call again with output plane descriptor buffer pointer (planeDesc.buf) in the XDM2_BufDesc structure.
27	JPEG_INVALID_OUTPUT_BUFFER_SIZE_ERROR	Output buffer size is zero	XDM_INSUFFICIENTDATA	Invoke process call again with proper output buffer sizes.
28	JPEG_INVALID_NUM_OF_OUTPUT_BUFFERS_ERROR	Number of output buffers passed to the codec is not 1	XDM_FATALERROR	Invoke process call again with proper number of output buffers.
29	JPEG_INSUFFICIENT_OUTPUT_BUFFER_SIZE_ERROR	Number of bytes encoded is greater than the number of bytes allocated to output buffer	XDM_INSUFFICIENTDATA	Invoke process call again with sufficient output buffer size.
30	JPEG_INVALID_JFIF_THUMBNAIL_ENABLE_ERROR	Invalid thumbnailIndexApp0 value (neither 0 nor 1) passed to codec through InArgs in process call	XDM_FATALERROR	Invoke process call again with proper thumbnailIndexApp0 value
31	JPEG_INVALID_EXIF_THUMBNAIL_ENABLE_ERROR	Invalid thumbnailIndexApp1 value (neither 0 nor 1) passed to codec through InArgs in process call	XDM_FATALERROR	Invoke process call again with proper thumbnailIndexApp1 value
32	JPEG_INPUT_BUFFER_POINTER_ALIGN_ERROR	The base address of the input 2D buffer in TILER region is not aligned to 16 bytes	XDM_FATALERROR	Invoke process call again with base address of the input 2D buffer in TILER region aligned to 16 bytes
33	JPEG_DATASYNC_GET_ROW_DATA_ERROR	numBlocks was set to a value less than 1 by input data sync call back function (getDataFxn)	No XDM Mapping	Invoke process call again after reviewing the implementation of the getDataFxn call back function
34	JPEG_DATASYNC_INVALID_RESTART_INTERVAL_ERROR	Invalid Restart Interval when Data Sync (Slice Mode) is Enabled	No XDM Mapping	Invoke process call again after enabling restart marker insertion
35	JPEG_DATASYNC_BLOCK_POINTER_ERROR	Invalid Buffer Pointer in the output data sync (getBufferFxn) call back function	No XDM Mapping	Invoke process call again after reviewing the implementation of the getBufferFxn call back function
36	JPEG_DATASYNC_BLOCK_SIZE_ERROR	Invalid Buffer Size in the output data sync (getBufferFxn) call back function	No XDM Mapping	Invoke process call again after reviewing the implementation of the getBufferFxn call back function
37	JPEG_DATASYNC_INVALID_BLOCKS_ERROR	Invalid Buffer Count in the output data sync (getBufferFxn) call back function	No XDM Mapping	Invoke process call again after reviewing the implementation of the getBufferFxn call back function
38	JPEG_DATASYNC_NOT_VALID_COMBINATION_ERROR	Invalid Combination of scatteredBlocksFlag and varBlockSizesFlag in the output data sync (getBufferFxn) call back function	No XDM Mapping	Invoke process call again after reviewing the implementation of the getBufferFxn call back function