MPEG4/H.263 Encoder on HDVICP2 and Media Controller Based Platform

User's Guide

Literature Number: SPRUGQ2
January 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	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
Wireless	www.ti.com/wireless-apps		

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 Instrument's (TI) MPEG4/H.263 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 and Media Controller 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,** provides answers to frequently asked questions related to using MPEG4 SP Encoder.
- **Appendix A - H.241 Packetization in case of H.263,** describes the mechanism that needs to be taken care by the application to handle the H.263 bitstream with H.241 packetization

- **Appendix B – Debug trace usage,** describes the debug trace tool supported by the codec and its usage.
- **Appendix C - DataSynch API Usage guide,** describes the data synch API usage from application point of view with codec.
- **Appendix D - Motion Vector and SAD Access API,** describes the method to access MV and SAD (Analytic Information) data provided by the encoder.
- **Appendix E – Picture Format,** describes the different format of uncompressed video, which are supported by encoder and the constraints

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 Interface 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.
- *Using IRES and RMAN Framework Components for C64x+* (literature number SPRAAI5), describes the IRES interface definition and function calling sequence.
- eXpressDSP Digital Media (XDM) Standard API Reference (literature number SPRUEC8)

Related Documentation

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

- ISO/IEC 14496-2:2003 Third Edition, Information Technology – Coding of Audio Visual Objects – Part 2 : Visual

Abbreviations

The following abbreviations are used in this document.

Table 1-1. List of Abbreviations

Abbreviation	Description
API	Application Programming Interface
SP	Simple Profile
CIF	Common Intermediate Format
COFF	Common Object File Format
DMA	Direct Memory Access
DMAN3	DMA Manager
DP	Data Partitioning
DSP	Digital Signal Processing
EVM	Evaluation Module
GOP	Group Of Pictures
GOB	Group of Blocks
HEC	Header Extension Code
HPI	Half Pixel Interpolation
IRES	Interface for Resources
PPS	Picture Parameter Set
QCIF	Quarter Common Intermediate Format
QP	Quantization Parameter
QVGA	Quarter Video Graphics Array
RMAN	Resource Manager
RVLC	Reversible Variable Length Coding
RM	Resync Marker
SQCIF	Sub Quarter Common Intermediate Format
VGA	Video Graphics Array
XDAIS	eXpressDSP Algorithm Interface Standard

Abbreviation	Description
XDM	eXpressDSP Digital Media

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 (MPEG4/H.263 Encoder on HDVICP2 and Media Controller Based Platform) and version number. The version number of the codec is included in the Title of the Release Notes that accompanies this codec.

Trademarks

Contents

Figures

This page is intentionally left blank

Tables

This page is intentionally left blank

Chapter 1

Introduction

This chapter provides a brief introduction to XDAIS and XDM interface. It also provides an overview of TI's implementation of the MPEG4/H.263 Encoder on the HDVICP2 and Media Controller Based Platform and its supported features.

Topic **Page**

1.1 Overview of XDAIS, XDM, and IRES

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). IRES is the interface for management and utilization of special resource types such as hardware accelerators, certain types of memory and DMA. This interface allows the client application to query and provide the algorithm its requested resources.

1.1.1 XDAIS Overview

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

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

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

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

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

1.1.2 XDM Overview

In the multimedia application space, you have the choice of integrating any codec into your multimedia system. For example, if you are building a video decoder system, you can use any of the available video decoders

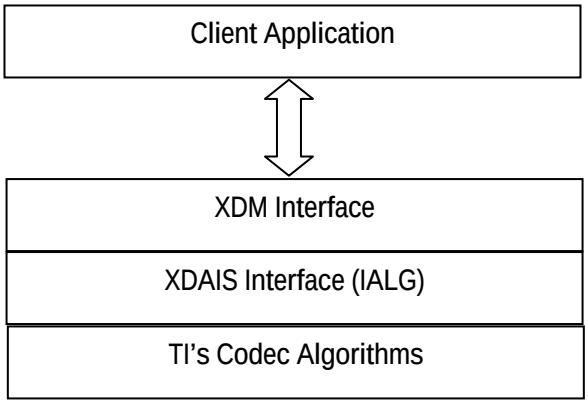
(such as MPEG4, H.263, or H.264) in your system. To enable easy integration with the client application, it is important that all codecs with similar functionality use similar APIs. XDM was primarily defined as an extension to XDAIS to ensure uniformity across different classes of codecs (for example 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 decoder, then you can easily replace MPEG4 with another XDM-compliant video decoder, say H.263, with minimal changes to the client application.

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

1.1.3 IRES Overview

IRES is a generic, resource-agnostic, extendible resource query, initialization and activation interface. The application framework defines, implements, and supports concrete resource interfaces in the form of IRES

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

The IRES interface introduces support for a new standard protocol for co-operative pre-emption, in addition to the IALG-style non-cooperative sharing of scratch resources. Co-operative pre-emption 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).

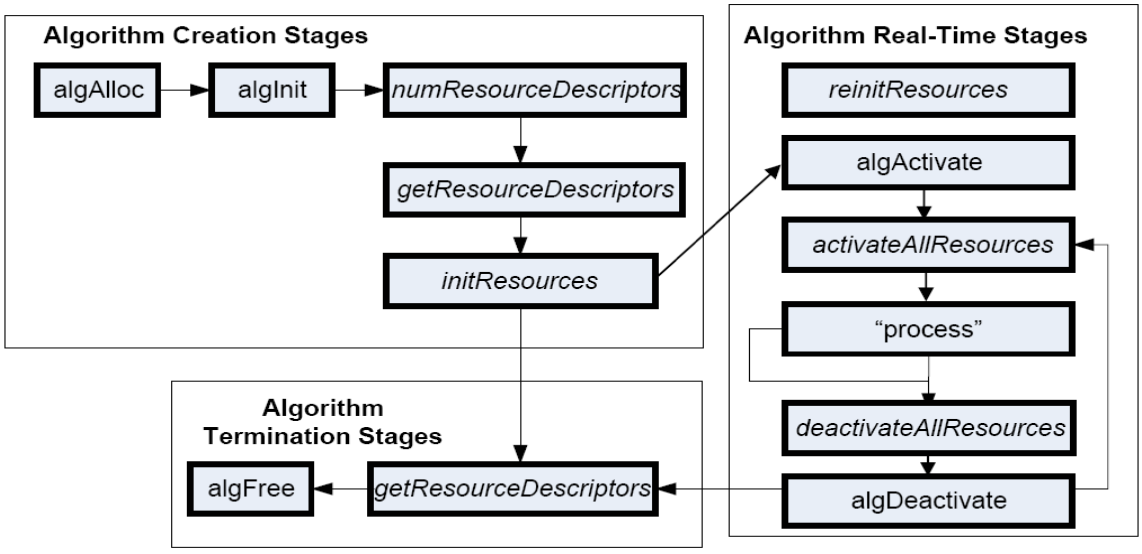


Figure 1-1. IRES Interface Definition and Function Calling Sequence.

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

1.2 Overview of MPEG4 SP Encoder

MPEG4(from ISO/IEC) is a popular video coding algorithm enabling high quality multimedia services on a limited bandwidth network. MPEG4 standard defines several profiles and levels that specify restrictions on the bit-stream and hence, limits the capabilities needed to encode/decode the bit-streams. Each profile specifies a subset of algorithmic features and limits encoders conforming to that profile. Each level specifies a set of limits on the values that can be taken by the syntax elements in the profile.

Some important features of MPEG4 Encoder are:

- Simple Profile:
  1. Only I and P type VOPs/Packets are present
  2. Only frame mode (progressive) picture types are present
  3. DP, RVLC and HEC support
  4. 1MV , 4MV and UMV support
  5. AC/DC prediction
  6. Motion Estimation and Compensation pixel accuracy up to half-pixel

Figure 1-2 depicts the working of the MPEG4 Encoder algorithm.

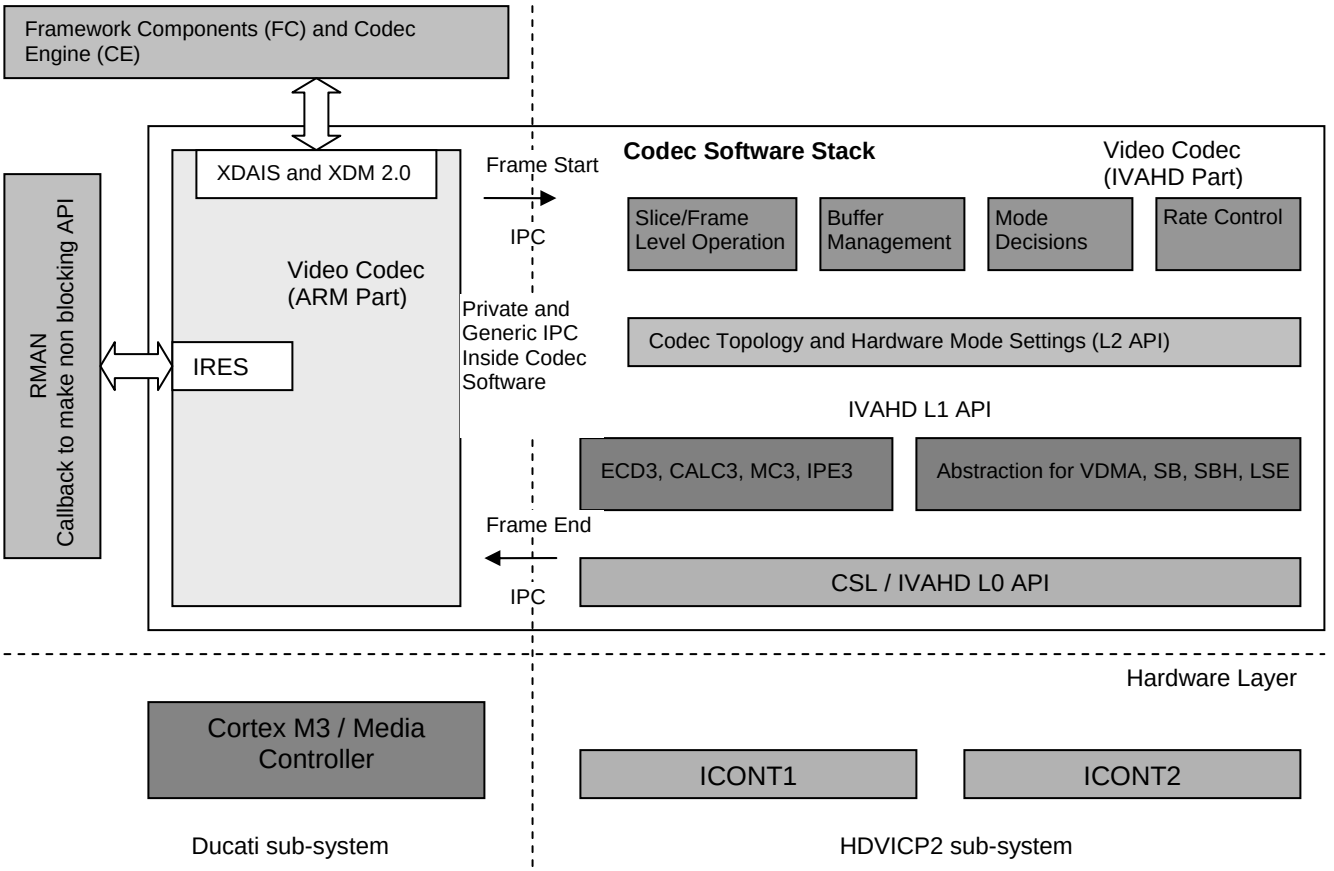


Figure 1-2. Working of MPEG4 Video Encoder

MPEG4/H.263 encoder implementation on HDVICP2 and Media Controller based platform has two parts:

- Core part of the encoding, which includes all frame and slice level operation and core-encoding algorithm. This part is implemented on HDVICP2 sub-system and the core part is hidden.
- Interface part of the encoder, which interacts with application and system software. This part is implemented on Media Controller. All the interfaces to query algorithm resource belongs to this part. This part of the video codec is exposed to system software.

Interface part of the video codec communicates with core part of video codec with private IPC defined in codec software through mailbox.

1.3 Supported Services and Features

This user guide accompanies TI's implementation of MPEG4/H.263 Encoder on the HDVICP2 and Media Controller Based Platform.

This version of the codec has the following features supported:

- MPEG4 Simple Profile levels 0, 0b, 1, 2, 3, 4A, 5 and 6 supported

- H.263 baseline profile levels 10, 20, 30, 40, 45, 50, 60 and 70 supported
- Only Progressive frame type picture encoding supported
- AC prediction supported
- Half-pel interpolation for motion estimation supported
- Unrestricted motion vector search that allows motion vectors to be outside the frame boundary is supported
- Custom picture format and GOB interval of H.263 is supported
- Resolution upto 2Kx2K supported
- 1MV/4MV per macro block is supported
- Supports low latency features - sub frame level synchronization for input data and bit-stream. Input Data synchronization is based upon MB row and output data synchronization is based upon slices and fixed length of bit-stream
- Encodes multiple slices per picture by inserting Resync Marker(RM), based on H.241 packetization or fixed number of macroblocks
- Rate control for low delay and storage applications
- Image width and height that are multiple of 16 are supported
- Supports Image height being non-multiple of 16 but multiple of 2
- Supports Image width being non-multiple of 16 but multiple of 2
- Supports user configurable Group of Pictures (GOP) length
- IDR frequency control is supported
- Supports different Intra Refresh mechanism
- Force I frame feature
- Scene change detection algorithm supported

The other explicit features that TI's MPEG4/H.263 Encoder provides are:

- eXpressDSP Digital Media (XDM IVIDENC2) interface compliant
- Supports booting of HDVICP2
- Implements Power Optimization schemes
- Supports YUV420 semi planar color sub-sampling formats
- Independent of any Operating System
- Ability to plug in any multimedia frameworks (For example, Codec engine, OpenMax, GStreamer etc.)

- Multi-channel functionality supported

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 is built and tested on HDVICP2 and Media Controller Based Platform.

2.1.2 Software

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

- **Development Environment:** This project is developed using Code Composer Studio (Code Composer Studio v4) version 4.2.0.09000. You may download Code Composer Studio v4 from the following location

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

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

Note:

Installing Code composer Studio v4 also installs CG tools version 4.5.1. However, it is recommended that you reinstall CG tools by downloading the latest version from the following location https://www-a.ti.com/downloads/sds_support/CodeGenerationTools.htm

The projects are built using g-make (GNU Make version 3.78.1)

- **Platform Simulator:** This project is developed using DM816x/OMAP4 Simulator. DM816x CSP version used is 0.7.1. This release of CSP version can be obtained via. software updates for CCSV4. 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_MPEG4SP_E_IVAHD_01_00, under which directory named IVAHD_001 is created.

Figure 2-1 shows the sub-directories created in the IVAHD_001directory.

Note:

The source folders under algsrc, icont and utils are not present in case of a library based (object code) release.

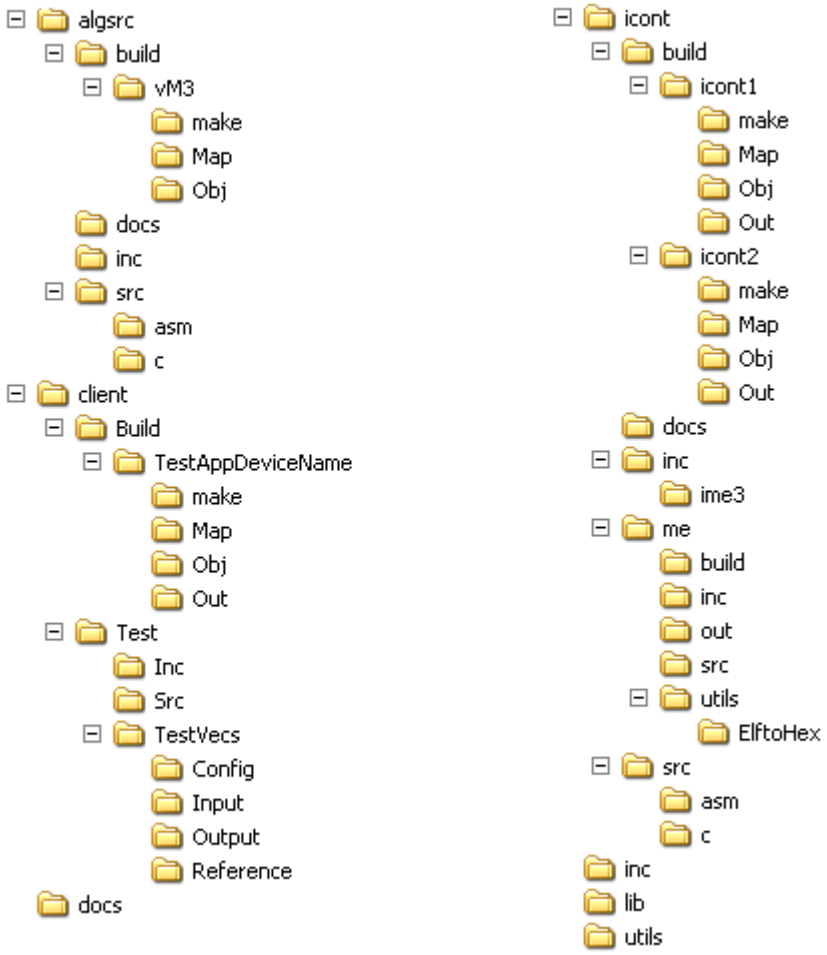


Figure 2-1. Component Directory Structure

Table 2-1 provides a description of the sub-directories created in the IVAHD_001 directory.

Table 2-1. Component Directories

Sub-Directory	Description
\algsrc\build\vM3\make	Contains the make file for application

Sub-Directory	Description
\algsrc\build\vM3\Obj	Contains intermediate Object files generated for Media Controller (host) project
\algsrc\docs	Contains documents specific to the Media Controller(host) project
\algsrc\inc	Contains header files needed by the Media Controller (host) project and some interface files which are shared between iCONT and Media Controller
\algsrc\src\asm	Contains assembly files needed by the Media Controller (host) project
\algsrc\src\c	Contains source files needed by the Media Controller (host) project
\Client\Build\TestAppDeviceName\ make	Contains the make file for application. The name of this directory will not be same as exactly mentioned here. Instead of DeviceName string, actual name of Device will be present
\Client\Build\TestAppDeviceName \Map	Contains the memory map generated on compilation of the code
\Client\Build\TestAppDeviceName \Obj	Contains the intermediate .asm and/or .obj file generated on compilation of the code
\Client\Build\TestAppDeviceName \Out	Contains the final application executable (.out) file generated by the sample test application
\Client\Test\Inc	Contains header files needed for the application code
\Client\Test\Src	Contains application C files
\Client\Test\TestVecs\Config	Contains sample configuration file for MPEG4 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 and datasheet
\icont\build\icont1\Map	Contains the generated map file related to icont1 project
\icont\build\icont1\make	Contains the make related to icont1 project
\icont\build\icont1\Obj	Contains the generated object files related to icont1 project
\icont\build\icont1\Out	Contains the generated executable file related to icont1 project
\icont\build\icont2\Map	Contains the generated map file related to icont2 project
\icont\build\icont2\make	Contains the make file related to icont2 project
\icont\build\icont2\Obj	Contains the generated object files related to icont2 project

Sub-Directory	Description
\icont\build\icont2\Out	Contains the generated executable file related to icont2 project
\icont\docs	Contains the iCONT module specific documents
\icont\inc	Contains the iCONT module specific header files
\icont\me\build	Contains project file related to Motion Estimation module
\icont\me\inc	Contains header file related to Motion Estimation module
\icont\me\out	Contains executable file related to Motion Estimation module
\icont\me\src	Contains source file related to Motion Estimation module
\icont\me\utils	Contains utility file(s) required by Motion Estimation module
\icont\src\asm	Contains assembly files needed by the iCONT1 and 2 projects
\icont\src\c	Contains source files needed by the iCONT1 and 2 projects
\Inc	Contains MPEG4 encoder related header files which allow interface to the codec library
\Lib	Contains the codec library file
\utils	Contains utility file(s) required by MPEG4 Encoder

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), HDVICP2 library and XDC tools.

This version of the codec has been validated on Framework Component (FC) version 3.20.00.22.

This version of the codec has been validated HDVICP2 library version 01.00.00.19.

2.3.1 Installing Framework Component (FC)

You can download the FC version 3.20.00.22 from the following location::

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 same location where you have installed Code Composer Studio. For example, if you have installed Code composer Studio at <install directory>\CCStudio_v4, extract the contents of zip file to <install directory>\CCStudio_v4.

Set a system environment variable named FC_INSTALL_DIR pointing to <install directory>\CCStudio4.0\<fc_directory>

The test application uses the following IRES and XDM files:

- HDVICP related IRES header files, these are available in the <install directory>\CCStudio_v4\<fc_directory>\packages \ti\sdo\fc\ires\hdvicp directory.
- Tiled memory related header file, these are available in the <install directory>\CCStudio_v4\<fc_directory>\fctools\packages \ti\sdo\fc\ires\tiledmemory directory.
- XDM related header files, these are available in the <install directory>\CCStudio_v4\<fc_directory>\fctools\packages \ti\xdais directory.
- Memutils file for memory address translation, these are available in the <install directory>\CCStudio_v4\<fc_directory>\fctools\packages \ti\ sdo\fc\memutils directory

2.3.2 Installing HDVICP2 library

Set a system environment variable named HDVICP2_INSTALL_DIR pointing to <hdvicp2_directory>\hdvicp20

Set a system environment variable named CSP_INSTALL_DIR pointing to <csp_directory>\csp.

The test application uses the HDVICP20 library file (ivahd_ti_api_vM3.lib) from <hdvicp2_directory>\hdvicp20\lib directory.

2.4 Building and Running the Sample Test Application on HDVICP2 and Media Controller based Platform

The sample test application that accompanies this codec component runs in TI's Code Composer Studio development environment. To build and run the sample test application in Code Composer Studio, follow these steps:

– Verify that you have an installation of TI's Code Composer Studio version 4.2.0.09000 and code generation tools version 4.5.1.

– Start the code composer studio and set up the target configuration for platform specific simulator / Emulator.

– Verify that the following codec object library exist in \Lib sub-directory:

  – mpeg4venc_ti_host.lib: MPEG4 encoder library

– Open the Code Composer Studio debug window with the appropriate platform configuration chosen.

– Build the sample test application project by gmake

  a) Client\Build\TestAppDeviceName> gmake -s deps

  b) Client\Build\TestAppDeviceName> gmake -k -s all

– All files required for this project are available at the path \Client\Build\TestAppDeviceName.

– The above step creates an executable file, TestAppEncoder.out in the \Client\Build\TestAppDeviceName\Out sub-directory

– Select **Target > Load Program** on Media Controller, browse to the \Client\Build\ TestAppDeviceName\Out sub-directory, select the codec executable created in step 5, and load it into Code Composer Studio in preparation for execution.

– If you are using sub-system simulator then make sure that iCONT1 and iCONT2 are in running state, even without loading any program. If you are using platform simulator then this step is not needed.

– Select **Target > Run** on Media Controller window to execute the sample test application.

The sample test application takes the input files stored in the \Client\Test\TestVecs\Input sub-directory, runs the codec. The reference files stored in the \Client\Test\TestVecs\Reference sub-directory can be used to verify that the codec is functioning as expected.

– On successful completion, the application displays the following messages for each frame:

"Frame# Frame Type FrameBits Across Process(MHz) Frame Based Mbps"

Note: MHz on simulator

As the simulator is not cycle accurate, the MHz across process() is printed to be zero.

– On failure, the application exits after encoding the frame in which codec failed to generate correct result.

2.5 Configuration Files

This codec is shipped along with:

- Encoder configuration file (encoder.cfg) – specifies the configuration parameters used by the test application to configure the Encoder.

2.5.1 Encoder Configuration File

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

A sample encoder.cfg file is as shown.

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

```
################################################################################
InputFile    = ..//..//..//Test//TestVecs//Input//airshow_p176x144_nv12.yuv"
EncodedFile  = "..//..//..//Test//TestVecs//Output//airshow_p176x144_nv12.m4v"
ReconFile    = "..//..//..//Test//TestVecs//Output//test_rec.yuv"
RefEncFile =
"..//..//..//Test//TestVecs//Reference//ref_airshow_p176x144_nv12.m4v"
FramesToBeEncoded          = 10     # Number of frame to be encoded
EncodingPreset             = 3     # 0 => XDM_DEFAULT 3=> XDM_USER_DEFINED
RateControlPreset          = 5     # 1 => Low Delay, 2 => Storage, 3 => 2
                                   # Pass, 4 => None, 5 => user defined
Level                      = 3     # Level IDC

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 = 9
InputContentType           = 0     # Progressive Type
OperatingMode              = 1     # Encode Mode
InputDataMode              = 3     # 2=> Input DataSync (Row Mode)
                                   # 3=> Process Entire Frame
NumInputUnits              = 5     # This Paramaeter will be used only when
                                   # InputDataMode = 0.
OutputDataMode             = 3     # 0=> Output DataSync (FIXEDLENGTH)
                                   # 1=> Output DataSync (SLICEMODE)
                                   # 3=> Encode entire frame into a bitstream
                                   # in single call.
NumOutputUnits             = 1     # This Paramaeter will be used only when
                                   # OutputDataMode = 0 or 1.
AspectRatioInfo            = 1     # If #15 need to send the par_width and
                                   #par_height
VopTimeIncrementResolution = 30

################################################################################
# Encoder Control
################################################################################
inputWidth                = 176    # Frame width should be multiple of 16
inputHeight               = 144    # Frame height
ReferenceFrameRate        = 30000
targetFrameRate           = 30000  # Target picture Rate per second *
                                   # 1000 => For 60 fields per second it
                                   # should be 60000
targetBitRate             = 128000 # Target Bit Rate in Bits per second.
intraFrameInterval        = 4      # Number of frames between two I frames.
MaxInterFrameInterval     = 0
CaptureWidth              = 176    # Image width to compute image pitch.
                                   # If CaptureWidth is > Image Width then use
                                   #the former for image pitch.
                                    # If CaptureWidth = 0, then inputWidth is
                                   #used as pitch
GenerateHeader            = 0      # Header is not encoded seperately. The
                                   #entire frame is encoded as an access
                                   # unit alongwith the headers.
ForceFrameType            = -1     # Frame type is not forced. It is as per
                                   #the encoding behaviour.
MotionVectorAccuracy      = 1      # HalfPel Accuracy.
SampleAspectRatioHeight   = 40     # Aspect Ratio Height => Need to have the
                                   #ImageHeight when aspect_ratio_info == 15
SampelAspectRatioWidth    = 33     # Aspect Ratio Width => Need to have the
                                   #ImageWidth when aspect_ratio_info == 15
IgnoreOutBufSizeFlag      = 0

################################################################################
# MPEG4 parameters
```

```
###############################################################################
NonMultipleOf16RefPadding = 1
UseHec                    = 0      # [0,2],
                                   # 0 => Do not use Header extension code
                                   # 1 => Use HEC only after first RM
                                   # 2 => Use HEC after all RMs
UseDataPartitioning       = 0
UseRvlc                   = 0
ShortVideoHeader          = 0      # 0 = MPEG-4, 1 = H.263 baseline
PixelRange                = 1      # video_range=0 : Y from 16 to 235,  Cb and
                                   #Cr from 16 to 240;
                                   # video_range=1 : Y from 0 to 255,Cb and Cr
                                   #from 0 to 255.
EnableSceneChangeAlgo     = 1      # 0 -> Disable, 1 -> Enable

###############################################################################
# InterCoding Control
###############################################################################
interCodingPreset         =   1    # 0 => deafult values, 1 => user defined
searchRangeHorP           =  44    # [16, 144]
searchRangeVerP           =  28    # [16, 32]
EarlySkipThreshold        = 200    # Threshold to use for early skip
                                   # determination
ThresholdingCost          =   1    # Thresholding cost used to set a block to
                                   # be not_coded
                                   # if the block has very few small amplitude
                                   # coeffs
InterSearch8x8            =  1     # Inter block search  8x8  (0=disable,
                                   # 1=enable)
GlobalOffsetME            =  1     # ME with global offset. 0 -> Disable, 1->
                                   # Enable
EnableRoundingControl     =  1     # When enabled alternatively toggles the
                                   # vop_rounding_type foe Inter frames
                                   # to reduce the IDCT drift
###############################################################################
# Intra Coding control
###############################################################################
AcPredEnable              = 1      # Enable AC prediction (0 = Off, 1 =
                                   # Enable)
intraCodingPreset         = 1      # 0 => deafult values, 1 => user defined
insertGOVHdrBeforeIframe  = 0      # [0,1] Inserts GOV Header before I frame
airMethod                 = 1      # Adaptive Intra Refresh method 0-> none 1-
                                   # > fixed pattern 2-> slice
                                   #  3-> MIR (Mandatory Intra Refresh)
airParam                  = 12     # if airMethod == 1, this variable holds
                                   # airCyclicMBPeriod,
                                   # if airMethod == 2, this variable holds
                                   # airCyclicRowWidth
EnableDriftControl        = 1      # Enable drift control (0 = Off, 1 =
                                   # Enable)

###############################################################################
# Rate control
###############################################################################
rateControlParamPreset    = 1      # 0 => Defualt; 1 => User Defined;
rcAlgo                    = 1      # 0 => FixedQP ; 1 => VBR ; 2 => CBR
rcFrameSkipEnable         = 0      # Enabling it makes it skip frame when
                                   # required
qpI                       = 5      # QP for I frame when no RC
qpP                       = 5      # QP for P frame when no RC
qpMax                     = 31
qpMin                     = 1
seIntialQP                = 5      # Initial QP used by Rate Control
```

```
PerceptualQuant           = 1      # Perceptual Quantization. 0 -> Disable, 1-
                                   # > Enable
VBVSize                   = 0      #0 -> Default value taken by codec 1 ->
                                   # User controlled
initialBufferLevel        = 0      #0 -> Default value taken by codec 1 ->
                                  # User controlled
qpMinIntra                = 0      # 0 – Disable, 1-31 -> Encoder tries to
                                   # encode Intra macroblocks in Inter frame
                                   # with qpMinIntra QP value
##############################################################################
#############
# Slice Mode Configuration
##############################################################################
#############
sliceCodingPreset         = 1      # 0 => deafult values, 1 => user defined
sliceMode                 = 0      # 0 => no Slices
                                   # 1 => Fixed MBs # 2 => Fixed Bits (RM)
sliceUnitSize             = 0      # if sliceMode = 1 => sliceUnitSize is
                                   # maxMbPerSlice
                                   # if sliceMode = 2 => sliceUnitSize is
                                   # resyncIntervalInBits or maxBitsPerSlice
GobInterval               = 1      # H.263 only, insert GOB header after every
                                   # n GOBs
##############################################################################
# Debug Trace Configuration
##############################################################################
DebugTraceLevel           = 0      # 0 -> off, 1 -> Level 1
LastNFramesToLog          = 0      # number of past frames – history

##############################################################################
# Misc
##############################################################################
ivahdId                   = 0      # Select ivahd id incase of multiple ivahd
                                   # present in SOC. E.g. DM816x
lumaTilerSpace            = 0      # Tiler enable for luma. 0-> No tiler
                                   # space, 1-> 8 bit
chromaTilerSpace          = 0      # Tiler enable for Chroma 0 -> No tiler
                                   # space , 1 -> 8 bit, 2-> 16 bit
```

Any field in the `IVIDENC2_Params` structure (see Section 4.2.1.6) can be set in the Encoder.cfg file using the syntax shown above. If you specify additional fields in the Encoder.cfg file, ensure to modify the test application appropriately to handle these fields.

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 codec for other input files of your choice, follow below steps:

1. Copy the input files to the \Client\Test\TestVecs\Inputs sub-directory
2. Edit the configuration file, Encoder.cfg available in the \Client\Test\TestVecs\Config sub-directory. For details on the format of the Encoder.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 MPEG4 SP Encoder library. The main test application files are MPEG4SPEncTest.c and mpeg4enc_ti_test.h. These files are available in the \Client\Test\Src and \Client\Test\Inc sub-directories respectively.

Figure 1-1 depicts the sequence of APIs exercised in the sample test application.

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 generic configuration file, configs.cfg and reads the various configuration parameters required for the algorithm
2. (Encoder.cfg), input file name, and output/reference file name.
3. Opens the Encoder configuration file, (Encoder.cfg) and reads the various configuration parameters required for the algorithm.
4. For more details on the configuration files, see Section 2.5.
5. Sets the IVIDENC2_Params structure based on the values it reads from the Encoder.cfg file.
6. Reads the input bit-stream into the application input buffer.

After successful completion of the above 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 implemented by the codec are called in sequence by `ALG_create()`:

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

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

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

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

After successful creation of the algorithm instance, the test application does resource allocation for the algorithm. This requires initialization of Resource Manager Module (RMAN) and grant of required resources (HDVICP2, Tiled memory, and so on) this is implemented by calling RMAN interface functions in following sequence:

- numResourceDescriptors() - To understand the number of resources (HDVICP2 and buffers) needed by algorithm.
  - getResourceDescriptors() - To get the attributes of the resources.
  - initResources() - After resources are created, application gives the resources to algorithm through this APII

3.1.3 Process Call

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

- Sets the dynamic parameters (if they change during run-time) by calling the control() function with the XDM_SETPARAMS command.
  - Sets the input and output buffer descriptors required for the process() function call. The input and output buffer descriptors are obtained by calling the control() function with the XDM_GETBUFINFO command.
  - Calls the process() function to encode/decode a single frame of data. The behavior of the algorithm can be controlled using various dynamic parameters (see Section 4.2.1.8). The inputs to the process function are input and output buffer descriptors, pointer to the IVIDENC2_InArgs and IVIDENC2_OutArgs structures.
  - When the process() function is called 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 placed in 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 parameters.

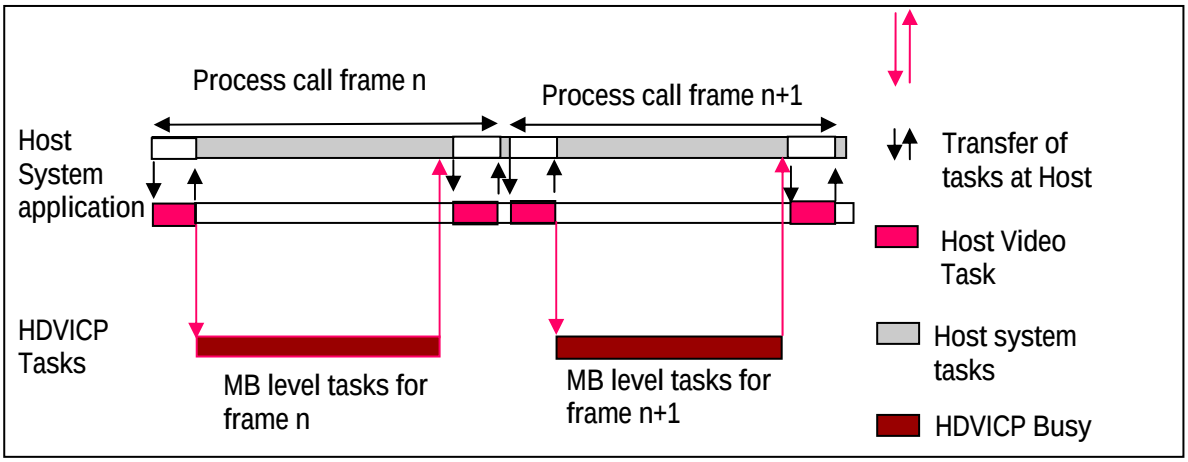


Figure 3-1. Process Call with Host Release

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. If the same algorithm is in-use between two process/control function calls, calling these functions can be avoided. Once an algorithm is activated, there can be any ordering of `control()` and `process()` functions. The following APIs are called in sequence:

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

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

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

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

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

The do-while loop encapsulates frame 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.

If the algorithm uses any resources through RMAN, then user must activate the resource after the algorithm is activated and deactivate the resource before algorithm deactivation.

3.1.4 Algorithm Instance Deletion

Once decoding/encoding is complete, the test application must release the resources granted by the IRES resource Manager interface and delete the current algorithm instance. The following APIs are called in sequence:

– `getResourceDescriptors()` - Free all resources granted by RMAN.

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

  – `algFree()` - To query the algorithm to get the memory record information.

  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.

  After successful execution of the algorithm, the test application frees up the memory and HDVICP Resource allocated for the algorithm. This is implemented by calling the RMAN interface functions in the following sequence:

– `RMAN_freeResources ()` - To free the resources that were allocated to the algorithm before process call.

  – `RMAN_unregister()` – To unregister the HDVICP protocol/resource manager with the generic resource manager.

  – `RMAN_exit()` - To delete the generic IRES RMAN and release memory.

3.2 Address Translations

The buffers addresses (DDR addresses) as seen by Media Controller and HDVICP2(VDMA) will be different. Hence, address translation is needed to convert from one address view to another. The application needs to implement a MEMUTILS function for this address translation. 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.

```
#define VDMAVIEW_EXTMEM (0xFFFFFFFF)
```

3.3 Handshaking Between Application and Algorithm

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

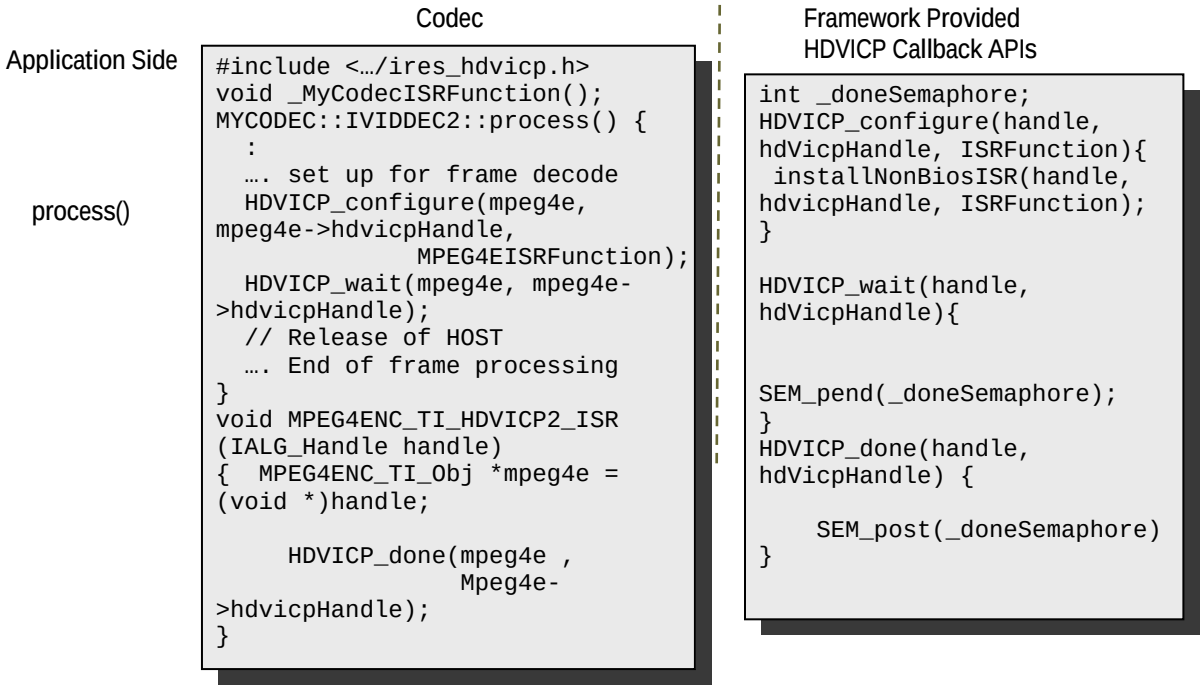


Figure 3-2. Interaction Between Application and Codec

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_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. The application needs to call this function, when it receives interrupts pertaining to the video task.

- `Void HDVICP_Acquire(IALG_Handle handle,IRES_HDVICP2_Handle iresHandle, IRES_YieldContext *yieldCtxt, IRES_HDVICP2_Status *status, Uint32* modeId, Int lateAcquireArg )`

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

- `Void HDVICP_Release(IALG_Handle handle, IRES_HDVICP2_Handle iresHandle)`

- This function is called by the algorithm to release the HDVICP2 resource.

- `Bool HDVICP_wait (void *hdvicpHandle)`

  This function is called by the algorithm to move the video task to SEM-`pend` state. Application should return false if it wants the early termination of codec.

- `Void 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.

- `Bool HDVICP_Reset(IALG_Handle handle, IRES_HDVICP2_Handle iresHandle)`

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

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 summarizes all the symbolic constants specified as either #define macros and/or enumerated C data types. For each symbolic constant, the semantics or interpretation is provided.

Table 4-1. List of Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IVIDEO_FrameType	For the various IVIDEO_xy_FRAME values, the frame type is interlaced where both top and bottom fields are provided in a single frame. The first field is an x frame, the second field is y field.	
	IVIDEO_NA_FRAME	Frame type not available
	IVIDEO_I_FRAME IVIDEO_FRAMETYPE_DEFAULT	Intra coded frame Default value
	IVIDEO_P_FRAME	Forward inter coded frame
	IVIDEO_B_FRAME	Bi-directional inter coded frame
	IVIDEO_IDR_FRAME	Intra coded frame that can be used for refreshing video content
	IVIDEO_II_FRAME	Interlaced frame, both fields are I frames
	IVIDEO_IP_FRAME	Interlaced frame, first field is an I frame, second field is a P frame
	IVIDEO_IB_FRAME	Interlaced frame, first field is an I frame, second field is a B frame
	IVIDEO_PI_FRAME	Interlaced frame, first field is a P frame, second field is a I frame
	IVIDEO_PP_FRAME	Interlaced frame, both fields are P frames.
	IVIDEO_PB_FRAME	Interlaced frame, first field is a P frame, second field is a B frame
	IVIDEO_BI_FRAME	Interlaced frame, first field is a B frame, second field is an I frame
	IVIDEO_BP_FRAME	Interlaced frame, first field is a B frame, second field is a P frame
	IVIDEO_BB_FRAME	Interlaced frame, both fields are B frames
	IVIDEO_MBAFF_I_FRAME	Intra coded MBAFF frame

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_MBAFF_P_FRAME	Forward inter coded MBAFF frame
	IVIDEO_MBAFF_B_FRAME	Bi-directional inter coded MBAFF frame
	IVIDEO_MBAFF_IDR_FRAME	Intra coded MBAFF frame that can be used for refreshing video content.
IVIDEO_ContentType	IVIDEO_CONTENTTYPE_NA	Frame type is not available
	IVIDEO_PROGRESSIVE IVIDEO_PROGRESSIVE_FRAME IVIDEO_CONTENTTYPE_DEFAULT	Progressive video content. Default value is IVIDEO_PROGRESSIVE
	IVIDEO_INTERLACED IVIDEO_INTERLACED_FRAME	Interlaced video content. Interlaced video content is not supported in MPEG4 Encoder
	IVIDEO_INTERLACED_TOPFIELD	Interlaced video content, top field
	IVIDEO_INTERLACED_BOTTOMFIELD	Interlaced video content, bottom field
IVIDEO_RateControlPreset	IVIDEO_LOW_DELAY	Constant Bit Rate (CBR) control for video conferencing
	IVIDEO_STORAGE IVIDEO_RATE_CONTROL_PRESET_DEFAULT	Variable Bit Rate (VBR) control for local storage (DVD) recording Default rate control preset value
	IVIDEO_TWOPASS	Two pass rate control for non-real time applications.
	IVIDEO_NONE	No configurable video rate control mechanism
	IVIDEO_USER_DEFINED	User defined configuration using extended parameters
IVIDEO_SkipMode	IVIDEO_FRAME_ENCODED IVIDEO_SKIPMODE_DEFAULT	Input video frame successfully encoded Default skip mode.
	IVIDEO_FRAME_SKIPPED	Input video frame skipped. There is no encoded bit-stream corresponding to the input frame.
IVIDEO_OutputFrameStatus	IVIDEO_FRAME_NOERROR IVIDEO_OUTPUTFRAMESTATUS_DEFAULT	Output buffer is available (default value) Default status of the output frame

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_FRAME_NOTAVAILABLE	Encoder does not have any output buffers.
	IVIDEO_FRAME_ERROR	Output buffer is available and corrupted. For example, if a bit-stream is erroneous and partially decoded, a portion of the decoded image may be available for display. Another example is if the bit-stream for a given frame decode may be decoded without error, but the previously decoded dependant frames were not successfully decoded. This would result in an incorrectly decoded frame. Not applicable for encoders.
IVIDEO_PictureType	IVIDEO_NA_PICTURE	Frame type not available
	IVIDEO_I_PICTURE IVIDEO_PICTURE_TYPE_DEFAULT	Intra coded picture Default value
	IVIDEO_P_PICTURE	Forward inter coded picture
	IVIDEO_B_PICTURE	Bi-directional inter coded picture
IVIDEO_Format	IVIDEO_MPEG1	Video format is MPEG1 stream
	IVIDEO_MPEG2SP	Video format is MPEG2/H.262 stream, Simple Profile
	IVIDEO_MPEG2MP	Video format is MPEG2/H.262 stream, Main Profile
	IVIDEO_MPEG2HP	Video format is MPEG2/H.262 stream, High Profile
	IVIDEO_MPEG4SP	Video format is MPEG4 stream, Simple Profile
	IVIDEO_MPEG4ASP	Video format is MPEG4 stream, Advanced Simple Profile
	IVIDEO_H264BP	Video format is H.264 stream, Base Profile
	IVIDEO_H264MP	Video format is H.264 stream, Main Profile
	IVIDEO_H264HP	Video format is H.264 stream, High Profile
	IVIDEO_VC1SP	Video format is VC1/WMV9 stream, Simple Profile

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_VC1MP	Video format is VC1/WMV9 stream, Main Profile
	IVIDEO_VC1AP	Video format is VC1 stream, Advanced Profile
	IVIDEO_H264RCDO	Video format is H.264 stream, Fast profile/RCDO
	IVIDEO_RV8	Video format is Real Video 8 stream
	IVIDEO_RV9	Video format is Real Video 9 stream
	IVIDEO_RV10	Video format is Real Video 10 stream, same as RV9
	IVIDEO_ON2VP6	Video format is ON2, VP6.x
	IVIDEO_ON2VP7	Video format is ON2, VP7.x
	IVIDEO_AVS10	Video format is AVS 1.0
	IVIDEO_SORENSONSPARK	Video format is SorensonSpark V0/V1
	IVIDEO_H263_PROFILE0	Video format is H263 Base line profile
	IVIDEO_H263_PROFILE3	Video format is H263 and Annex IJKT
	IVIDEO_H264SVC	Video format is SVC
	IVIDEO_MULTIVIEW	Video format is multi-view coding
IVIDEO_VideoLayout	IVIDEO_FIELD_INTERLEAVED	Buffer layout is interleaved
	IVIDEO_FIELD_SEPARATED	Buffer layout is field separated
	IVIDEO_TOP_ONLY	Buffer contains only top field
	IVIDEO_BOTTOM_ONLY	Buffer contains only bottom field
IVIDEO_OperatingMode	IVIDEO_DECODE_ONLY	Decoding mode Not applicable for encoders
	IVIDEO_ENCODE_ONLY	Encoding mode
	IVIDEO_TRANSCODE_FRAMELEVEL	Transcode mode of operation (encode/decode) that consumes/generates transcode information at the frame level

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_TRANSCODE_MBLEVEL	Transcode mode of operation (encode/decode) that consumes/generates transcode information at the MB level
	IVIDEO_TRANSRATE_FRAMELEVEL	Transrate mode of operation for encoder that consumes transrate information at the frame level
	IVIDEO_TRANSRATE_MBLEVEL	Transrate mode of operation for encoder which consumes transrate information at the MB level
		Not supported in this version of MPEG4 Encoder
1) IVIDEO_BitRange	IVIDEO_YUVRANGE_FULL	Pixel range for YUV is 0-255
	IVIDEO_YUVRANGE_ITU	Pixel range for YUV is as per ITU-T
2) IVIDEO_DataMode	IVIDEO_FIXEDLENGTH	Data is exchanged at interval of fixed size
	IVIDEO_SLICEMODE	Slice mode
	IVIDEO_NUMROWS	Number of rows, each row is 16 lines of video
	IVIDEO_ENTIREFRAME	Processing of entire frame data
3) IVIDEO_ErrorInfoMode	IVIDEO_ERRORINFO_OFF IVIDEO_ERRORINFO_MODE_DEFAULT	Packet error information is unsupported Default error info mode.
	IVIDEO_ERRORINFO_ON_INPUT	Packet error information is supported for input data
	IVIDEO_ERRORINFO_ON_OUTPUT	Packet error information is supported for output data
	IVIDEO_ERRORINFO_ON_BOTH	Packet error information is supported for both input and output data
4) XDM_AccessMode	XDM_ACCESSMODE_READ	Algorithm read from the buffer using the CPU
	XDM_ACCESSMODE_WRITE	Algorithm writes to the buffer using the CPU
5) XDM_CmdId	XDM_GETSTATUS	Query algorithm instance to fill `Status` structure

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_SETPARAMS	Set run-time dynamic parameters through the DynamicParams structure
	XDM_RESET	Reset the algorithm. All fields in the internal data structures are reset and all internal buffers are flushed.
	XDM_SETDEFAULT	Restore the algorithm's internal state to its original, default values. The application only needs to initialize the dynamicParams.size and status.size fields prior to calling control() with XDM_SETDEFAULT. The algorithm must only write to the status.extendedError field, and potentially algorithm specific, extended fields. XDM_SETDEFAULT differs from XDM_RESET. In addition to restoring the algorithm's internal state, XDM_RESET also resets any channel related state.
	XDM_FLUSH	Handle end of stream conditions. This command forces the algorithm to output data without additional input. The recommended sequence is to call the control() function (with XDM_FLUSH) followed by repeated calls to the process() function until it returns an error. The algorithm should return the appropriate, class-specific EFAIL error (example, ISPHDEC1_EFAIL, IVIDENC1_EFAIL, and so on), when flushing is complete.
	XDM_GETBUFINFO	Query algorithm instance regarding its properties of input and output buffers. The application only needs to initialize the dynamicParams.size, the status.size, and set any buffer descriptor fields (example, status.data) to NULL prior to calling control() with XDM_GETBUFINFO.
	XDM_GETVERSION	Query the algorithm's version. The result is returned in the data field of the respective Status structure. There is no specific format defined for version returned by the algorithm.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_GETCONTEXTINFO	Query a split codec part for its context needs. Only split codecs are required to implement this command. Not supported in this version of MPEG4 Encoder.
	XDM_GETDYNPARAMSDEFAULT	Query the algorithm to fill the default values for the parameters which can be configured dynamically. To get the current value of an algorithm instance's dynamic parameters, it is recommended that the algorithm provides them through the XDM_GETSTATUS call. Some XDM interfaces provide a struct in the base class Status structure in which the current state can be returned (example, IVIDENC2_Status.encDynamicParams), other codec classes do not (and algorithms could) provide this through an extended Status structure.
6) XDM_DataFormat	XDM_BYTE	Big endian stream (default value)
	XDM_LE_16	16-bit little endian stream.
	XDM_LE_32	32-bit little endian stream.
	XDM_LE_64	64-bit little endian stream.
	XDM_BE_16	16-bit big endian stream.
	XDM_BE_32	32-bit big endian stream.
	XDM_BE_64	64-bit big endian stream.
XDM_ChromaFormat	XDM_CHROMA_NA	Chroma format not applicable
	XDM_YUV_420P	YUV 4:2:0 planar.
	XDM_YUV_422P	YUV 4:2:2 planar.
	XDM_YUV_422IBE	YUV 4:2:2 interleaved (big endian).
	XDM_YUV_422ILE	YUV 4:2:2 interleaved (little endian)
	XDM_YUV_444P	YUV 4:4:4 planar.
	XDM_YUV_411P	YUV 4:1:1 planar.
	XDM_GRAY	Gray format.
	XDM_RGB	RGB color format.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_YUV_420SP	YUV 4:2:0 chroma semi-planar format (first plane is luma and second plane is CbCr interleaved) Default value.
	XDM_ARGB8888	Alpha plane color format.
	XDM_RGB555	RGB555 color format.
	XDM_RGB565	RGB565 color format.
	XDM_YUV_444ILE	YUV 4:4:4 interleaved (little endian) color format.
XDM_MemoryType	XDM_MEMTYPE_ROW XDM_MEMTYPE_RAW	Raw memory type
	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
	XDM_MEMTYPE_TILEDPAGE	2D memory in page container of tiled memory space
XDM_EncodingPreset	XDM_DEFAULT	Default setting of the algorithm specific creation time parameters
	XDM_HIGH_QUALITY	Set algorithm specific creation time parameters for high quality.
	XDM_HIGH_SPEED	Set algorithm specific creation time parameters for high speed.
	XDM_USER_DEFINED XDM_PRESET_DEFAULT	User defined configuration using advanced parameters Default value
	XDM_HIGH_SPEED_MED_QUALITY	Set algorithm specific creation time parameters for high speed medium quality
	XDM_MED_SPEED_MED_QUALITY	Set algorithm specific creation time parameters for medium speed medium quality.
	XDM_MED_SPEED_HIGH_QUALITY	Set algorithm specific creation time parameters for medium speed high quality
XDM_EncMode	XDM_ENCODE_AU	Encode entire access unit, including the headers (default value)

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_GENERATE_HEADER	Encode only header
IVIDENC2_MotionVectorAccuracy	IVIDENC2_MOTIONVECTOR_PIXEL	Motion vectors accuracy is only integer pel Not supported in this version of MPEG4 encoder
	IVIDENC2_MOTIONVECTOR_HALFPEL	Motion vectors accuracy is half pel
	IVIDENC2_MOTIONVECTOR_QUARTERPEL	Motion vectors accuracy is quarter pel Not supported in this version of MPEG4 encoder
	IVIDENC2_MOTIONVECTOR_EIGHTHPEL	Motion vectors accuracy is one-eighth pel Not supported in this version of MPEG4 encoder
	IVIDENC2_MOTIONVECTOR_MAX	Motion vectors accuracy is not defined Not supported in this version of MPEG4 encoder
XDM_ErrorBit	XDM_PARAMSCHANGE	Bit 8 • 1 - Sequence Parameters Change • 0 - Ignore This error is applicable for transcoders. It is set when some key parameter of the input sequence changes. The transcoder returns after setting this error field and the correct input sequence parameters are updated in `outArgs`.
	XDM_APPLIEDCONCEALMENT	Bit 9 • 1 - Applied concealment • 0 - Ignore This error is applicable for decoders. It is set when the decoder was not able to able to decode the bit-stream, and the decoder has concealed the bit-stream error and produced the concealed output.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_INSUFFICIENTDATA	Bit 10 • 1 - Insufficient input data • 0 - Ignore This error is typically applicable for decoders. This is set when the input data provided is not sufficient to produce of one frame of data. This can be also be set for encoders when the number of valid samples in the input frame is not sufficient to process a frame.
	XDM_CORRUPTEDDATA	Bit 11 • 1 - Data problem/corruption • 0 - Ignore This error is typically applicable for decoders. This is set when the bit-stream has an error and not compliant to the standard syntax.
	XDM_CORRUPTEDHEADER	Bit 12 • 1 - Header problem/corruption • 0 - Ignore This error is typically applicable for decoders. This is set when the header information in the bit-stream is incorrect. For example, it is set when Sequence, Picture, Slice, and so on are incorrect in video decoders.
	XDM_UNSUPPORTEDINPUT	Bit 13 • 1 - Unsupported feature/parameter in input • 0 - Ignore This error is set when the algorithm is not able process a certain input data/bit-stream format. It can also be set when a subset of features in a standard are not supported by the algorithm. For example, if a video encoder only supports 4:2:2 formats, it can set this error for any other type of input video format.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_UNSUPPORTEDPARAM	Bit 14 • 1 - Unsupported input parameter or configuration • 0 - Ignore This error is set when the algorithm does not support certain configurable parameters. For example, if the video decoder does not support the display width feature, it will return XDM_UNSUPPORTEDPARAM when the control function is called for setting the display width attribute.
	XDM_FATALERROR	Bit 15 • 1 - Fatal error (stop encoding) • 0 - Recoverable error If there is an error and this bit is not set, the error is recoverable. This error is set when the algorithm cannot recover from the current state. It informs the system not to try the next frame and possibly delete the multimedia algorithm instance. It implies the codec will not work when reset. You should delete the current instance of the codec.

Note:

The remaining bits that are not mentioned in XDM_ErrorBit are interpreted as:

- Bit 16-32: Reserved
- Bit 0-7: Codec and implementation specific (see Table 4-3)

The algorithm can set multiple bits to one depending on the error condition.

Table 4-2. MPEG4 Encoder Specific Enumerated Data Types.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IMPEG4VENC_Level	IMP4VENC_SP_LEVEL_0	MPEG-4 Simple profile level 0 Value = 0
	IMP4VENC_SP_LEVEL_0B	MPEG-4 Simple profile level 0b Value = 9

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMP4VENC_SP_LEVEL_1	MPEG-4 Simple profile level 1 Value = 1
	IMP4VENC_SP_LEVEL_2	MPEG-4 Simple profile level 2 Value = 2
	IMP4VENC_SP_LEVEL_3	MPEG-4 Simple profile level 3 Value = 3
	IMP4VENC_SP_LEVEL_4A	MPEG-4 Simple profile level 4a Value = 4
	IMP4VENC_SP_LEVEL_5	MPEG-4 Simple profile level 5 Value = 5
	IMP4VENC_SP_LEVEL_6	MPEG-4 Simple profile level 5 Value = 6
IMPEG4ENC_H263Level	Level information for H263 Baseline	
	IMPEG4ENC_H263_LEVEL_10	H263 baseline profile level 10
	IMPEG4ENC_H263_LEVEL_20	H263 baseline profile level 20
	IMPEG4VENC_H263_LEVEL_30	H263 baseline profile level 30
	IMPEG4VENC_H263_LEVEL_40	H263 baseline profile level 40
	IMPEG4VENC_H263_LEVEL_45	H263 baseline profile level 45
	IMPEG4VENC_H263_LEVEL_50	H263 baseline profile level 50
	IMPEG4VENC_H263_LEVEL_60	H263 baseline profile level 60
	IMPEG4VENC_H263_LEVEL_70	H263 baseline profile level 70
IMPEG4ENC_nonMultiple16RefPadMethod	These enumerations define the type of padding done when dimension is non-multiple of 16.	
	IMPEG4_PAD_METHOD_DIVX	Method as suggested by DivX spec.
	IMPEG4_PAD_METHOD_MPEG4	Method as suggested by MPEG4 spec.
	IMPEG4_PAD_METHOD_DEFAULT	Takes the values of IMPEG4_PAD_METHOD_MPEG4
IMPEG4ENC_InterCodingPreset	These enumerations control the type of inter coding.	
	IMPEG4_INTERCODING_DEFAULT	Default inter coding params
	IMPEG4_INTERCODING_USERDEFINED	User defined inter coding params Default value

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG4_INTERCODING_EXISTING	Keep inter coding params as existing. This is useful during control call, if you do not want to change the inter coding params
IMPEG4ENC_InterBlockSize	These enumerations are defined for minimum inter block size.	
	IMPEG4_BLOCKSIZE_16x16	16x16 block size
	IMPEG4_BLOCKSIZE_8x8	8x8 block size
	IMPEG4_BLOCKSIZE_DEFAULT	Default block size
IMPEG4ENC_RateControlAlgoPreset	These enumerations control the RateControl Algorithm	
	IMPEG4_RATECONTROLALGO_NONE	No RC. Fixed QP Encoding.
	IMPEG4_RATECONTROLALGO_VBR	VBR RC Algorithm
	IMPEG4_RATECONTROLALGO_CBR	CBR RC Algorithm aka low Delay RC Algorithm
IMPEG4ENC_IntraCodingPreset	These enumerations control the type of intra coding.	
	IMPEG4_INTRACODING_DEFAULT	Default intra coding params
	IMPEG4_INTRACODING_USERDEFINED	User defined intra coding params Default value
IMPEG4ENC_IntraRefreshMethods	Intra refresh method type identifier for MPEG4 Encoder	
	IMPEG4_INTRAREFRESH_NONE	Do not insert forcefully any intra macro blocks
	IMPEG4_INTRAREFRESH_CYCLIC_MBS	Inserts intra macro blocks in a cyclic fashion . Cyclic interval is equal to intraRefreshRate
	IMPEG4_INTRAREFRESH_CYCLIC_ROWS	Inserts Intra Rows in a cyclic fashion. Cyclic interval is equal to intraRefreshRate.
	IMPEG4_INTRAREFRESH_MANDATORY	Mandatory Intra Refresh – Inserts Intra MBs sucn that it evenly distributes number of INTRA MBs over frames.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG4_INTRAREFRESH_RDOPT_MBS	Position of intra macro blocks is chosen by encoder, but the number of forcibly coded intra macro blocks in a frame is gaurnteed to be equal to totalMbsInFrame/ intraRefreshRate. This method is not implemented currently.
IMPEG4ENC_SliceCodingPreset	These enumerations control the slice coding parameters	
	IMPEG4_SLICECODING_DEFAULT	Default slice coding params
	IMPEG4_SLICECODING_USERDEFINED	User defined slice coding params Default value
	IMPEG4_SLICECODING_EXISTING	Keep slice coding params as existing. This is useful during control call, if you do not want to change the slice coding parameters.
IMPEG4ENC_SliceMode	These enumerations captures the slice coding mechanism	
	IMPEG4_SLICEMODE_NONE	Does not enable any Slice Mechanism
	IMPEG4_SLICEMODE_MBUNIT	Enables the Slice Coding Mechanism based on number of Macroblocks
	IMPEG4_SLICEMODE_BITS	Enables the Slice Coding Mechanism based on number of bits consumed. H.241 Mechanism.
IMPEG4ENC_PixelRange	IMPEG4ENC_PR_16_235	video_range=0, gives a range of Y from 16 to 235, Cb and Cr from 16 to 240. See Section 6.3.2 in MPEG-4 visual standard
	IMPEG4ENC_PR_0_255	video_range=1 gives a range of Y from 0 to 255, Cb and Cr from 0 to 255. See Section 6.3.2 in MPEG-4 visual standard.
	IMPEG4ENC_PR_DEFAULT	IMPEG4ENC_PR_0_255
IMPEG4ENC_SceneChangeAlgo	IMPEG4ENC_SCDA_DISABLE	Disables the algorithm to detect scene change in the video sequence
	IMPEG4ENC_SCDA_ENABLE	Enables the algorithm to detect scene change in the video sequence.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG4ENC_SCDA_DEFAULT	IMPEG4ENC_SCDA_ENABLE
IMPEG4ENC_AspectRatioIdc	IMPEG4ENC_ASPECTRATIO_SQUARE	1:1 (square) aspect ratio
	IMPEG4ENC_ASPECTRATIO_12_11	12:11 aspect ratio
	IMPEG4ENC_ASPECTRATIO_10_11	10:11 aspect ratio
	IMPEG4ENC_ASPECTRATIO_16_11	16:11 aspect ratio
	IMPEG4ENC_ASPECTRATIO_40_33	40:33 aspect ratio
	IMPEG4ENC_ASPECTRATIO_EXTENDED	Extended Aspect Ratio

The MPEG4 SP Encoder specific error status messages are listed in Table 4-3.

Table 4-3. MPEG4 Encoder Error Statuses

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IMPEG4VENC_ErrorBit	IMPEG4ENC_LEVEL_INCOMPLAINT_PARAMETER	Bit 0 – Level in-complaint parameter This error is thrown when the resolution/frame rate/bit rate exceeds the standard specified values for the level set by the application.
	IMPEG4ENC_PROFILE_INCOMPLAINT_CONTENTTYPE	Bit 1 - Profile in-complaint content type. This error is applicable when IVIDENC2_Params::inputContentType is not set as IVIDEO_PROGRESSIVE , and IVIDENC2_Params::profile is set as IMPEG4_BASELINE_PROFILE.
	IMPEG4ENC_IMPROPER_HDVICP2_STATE	Bit 16 – HDVCIP2 is not in correct state. Before using the HDVICP2, the encoder checks clock setting for all the modules of HDVICP2 and checks for HDVCIP2 being in standby state. If not then codec throws this error

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG4ENC_WARNING_H263_PLUS_CUSTOM_SOURCE_FORMAT	Bit 17 - Indication that the input resolution given to codec is custom source format supported in H.263+ not the standard resolutions supported with H263 baseline or MPEG4 with short video header.
	IMPEG4ENC_ERROR_BITSTREAM_MEMORY_INSUFFICIENT	Bit 18 - Indication that the buffer given to codec from getBuffer function is insufficient so that codec cannot continue encoding. It means that if return value from getBuffer function is -1, then this bit gets set by the codec. This is the situation where application might not be able to provide memory to codec.
	IMPEG4ENC_IMPROPER_DATASYNC_SETTING	Bit 19 - data synch settings are not proper. This error is set when encoder is asked to operate at sub frame level but the call back function pointer is NULL.
	IMPEG4ENC_UNSUPPORTED_VIDENC2PARAMS	Bit 20 - Invalid videnc2 parameters. This error is set when any parameter of structure IVIDENC2_Params is not in allowed range.
	IMPEG4ENC_UNSUPPORTED_RATECONTROLPARAMS	Bit 21 - Invalid rate control parameters. This error is set when any parameter of structure IMPEG4ENC_RateControlParams is not in allowed range.
	IMPEG4ENC_UNSUPPORTED_INTERCODINGPARAMS	Bit 22 - Invalid inter coding parameters. This error is set when any parameter of structure IMPEG4ENC_InterCodingParams is not in allowed range.
	IMPEG4ENC_UNSUPPORTED_INTRACODINGPARAMS	Bit 23 - Invalid Intra coding parameters. This error is set when any parameter of structure IMPEG4ENC_IntraCodingParams is not in allowed range.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG4ENC_UNSUPPORTED_SLICECODINGPARAMS	Bit 25 - Invalid slice coding parameters This error is set when any parameter of structure IMPEG4ENC_SliceCodingParams is not in allowed range
	IMPEG4ENC_UNSUPPORTED_MPEG4ENCPARAMS	Bit 29 - Invalid Create time extended parameters This error is set when any parameter of structure IMPEG4ENC_Params is not in allowed range
	IMPEG4ENC_UNSUPPORTED_VIDENC2DYNAMICPARAMS	Bit 30 - Invalid base class dynamic parameters during control This error is set when any parameter of structure IVIDENC2_DynamicParams is not in allowed range
	IMPEG4ENC_UNSUPPORTED_MPEG4ENCDYNAMICPARAMS	Bit 31 - Invalid extended class dynamic parameters during control This error is set when any parameter of structure IMPEG4ENC_DynamicParams (excluding embedded structures) is not in allowed range

4.2 Data Structures

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.

4.2.1 Common XDM Data Structures

This section includes the following common XDM data structures:

- `XDM2_SingleBufDesc`
- `XDM2_BufDesc`
- `XDM1_AlgBufInfo`
- `IVIDEO1_BufDescIn`
- `IVIDEO2_BufDesc`
- `IVIDENC2_Fxns`
- `IVIDENC2_Params`
- `IVIDENC2_DynamicParams`
- `IVIDENC2_Inargs`
- `IVIDENC2_Status`
- `IVIDENC2_OutArgs`
- `XDM_Date`
- `XDM_Point`
- `XDM_Rect`
- `XDM_DataSyncDesc`

4.2.1.1 XDM2_SingleBufDesc

Description

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

Fields

Field	Data Type	Input/ Output	Description
*buf	XDAS_Int8	Input	Pointer to the buffer address
memType	XDAS_Int32	Input	Type of memory, See XDM_MemoryType enumeration in Table 4-1 for more details
bufSize	XDM2_BufSize	Input	Buffer size for tile memory/row memory
accessMask	XDAS_Int32	Input	Mask filled by the algorithm, declaring how the buffer was accessed by the algorithm processor. If the buffer was not accessed by the algorithm processor (example, it was filled through DMA or other hardware accelerator that does not write through the algorithm's CPU), then bits in this mask should not be set. It is acceptable to set several bits in this mask, if the algorithm accessed the buffer in several ways. This mask is often used by the application and/or framework to manage cache on cache-based systems. See XDM_AccessMode enumeration in Table 4-1 for more details. In the current version of the MPEG4 Encoder, Media Controller does not read or write the buffers given by out by application meaning accessMask for inBufs or outBufs is zero.

4.2.1.2 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. Must be less than XDM_MAX_IO_BUFFERS.
descs[XDM_MAX_IO_BUFFERS]	XDM2_SingleBufDesc	Input	Array of buffer descriptors

4.2.1.3 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	Minimum number of input buffers
`minNumOutBufs`	`XDAS_Int32`	Output	Minimum number of output buffers
`minInBufSize[XDM_MAX_IO_BUFFERS]`	`XDM2_BufSize`	Output	Minimum size required for each input buffer
`minOutBufSize[XDM_MAX_IO_BUFFERS]`	`XDM2_BufSize`	Output	Minimum size required for each output buffer
`inBufMemoryType[XDM_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	Required memory type for each input buffer. See `XDM_MemoryType` enumeration in Table 4-1 for more details.
`outBufMemoryType[XDM_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	Required memory type for each output buffer. See `XDM_MemoryType` enumeration in Table 4-1 for more details.
`minNumBufSets`	`XDAS_Int32`	Output	Minimum number of buffer sets for buffer management

Note:

For MPEG4 Encoder, the buffer details are:

- Number of input buffer required is 2 for YUV 420SP chroma format (`memType` is `XDM_MEMTYPE_TILED8` and `XDM_MEMTYPE_TILED16`)
- Number of output buffer required is 1 (`memType` is `XDM_MEMTYPE_ROW and XDM_MEMTYPE_TILEDPAGE`)
- The input buffer sizes (in bytes) for CIF format is:

  Y buffer = 352 * 288
  UV buffer = 352* 144

- There is no restriction on output buffer size except that it should contain atleast one frame of encoded data.

These are the example buffer sizes but you can reconfigure depending on the input format.

4.2.1.4 IVIDEO1_BufDescIn

Desciption

This structure defines the buffer descriptor for inputs video buffers.

‖ Fields

Field	Data Type	Input/ Output	Description
numBufs	XDAS_Int32	Input	Number of buffers in bufDesc[]
frameWidth	XDAS_Int32	Input	Width of the video frame
frameHeight	XDAS_Int32	Input	Height of the video frame
framePitch	XDAS_Int32	Input	Frame pitch used to store the frame. This field can also be used to indicate the padded width.
bufDesc[XDM_MAX_IO_BUFFERS]	XDM1_SingleBufDesc	Input	Picture buffers.

4.2.1.5 IVIDEO2_BufDesc

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
numPlanes	XDAS_Int32	Input/Output	Number of buffers for video planes
numMetaPlanes	XDAS_Int32	Input/Output	Number of buffers for Metadata
dataLayout	XDAS_Int32	Input/Output	Video buffer layout, field interleaved or field separated. See IVIDEO_VideoLayout enumeration in Table 4-1for more details
planeDesc [IVIDEO_MAX_NUM_PLANES]	XDM2_SingleBufDesc	Input/Output	Description for video planes
metadataPlaneDesc [IVIDEO_MAX_NUM_METADATA_PLANES]	XDM2_SingleBufDesc	Input/Output	Description for metadata planes
secondFieldOffsetWidth[IVIDEO_MAX_NUM_PLANES]	XDAS_Int32	Input/Output	Offset value for second field in planeDesc buffer (width in pixels) Valid only if pointer is not NULL.
secondFieldOffsetHeight[IVIDEO_MAX_NUM_PLANES]	XDAS_Int32	Input/Output	Offset value for second field in planeDesc buffer (height in lines) Valid only if pointer is not NULL.
imagePitch	XDAS_Int32	Input/Output	Image pitch, common for all planes

Field	Data Type	Input/ Output	Description
imageRegion	XDM_Rect	Input/Output	Decoded image region including padding/encoder input image (top left and bottom right).
activeFrameRegion	XDM_Rect	Input/Output	Actual display region/capture region top left and bottom right).
extendedError	XDAS_Int32	Input/Output	Indicates the error type, if any. Not applicable for encoders.
frameType	XDAS_Int32	Input/Output	Video frame types. See enumeration IVIDEO_FrameType enumeration in Table 4-1 for more details. Not applicable for encoder input buffer.
topFieldFirstFlag	XDAS_Int32	Input/Output	Indicates when the application (should display)/(had captured) the top field first. Not applicable for progressive content. Not applicable for encoder reconstructed buffers. Valid values are XDAS_TRUE and XDAS_FALSE.
repeatFirstFieldFlag	XDAS_Int32	Input/Output	Indicates when the first field should be repeated. Valid values are XDAS_TRUE and XDAS_FALSE. Only applicable for interlaced content, not progressive. Not applicable for encoders.
frameStatus	XDAS_Int32	Input/Output	Video in/out buffer status. Not applicable for encoder reconstructed buffers.
repeatFrame	XDAS_Int32	Input/Output	Number of times the display process needs to repeat the displayed progressive frame. This information is useful for progressive content when the decoder expects the display process to repeat the displayed frame for a certain number of times. This is useful for pull-down (frame/field repetition by display system) support where the display frame rate is increased without increasing the decode frame rate. Default value is 0. Not applicable for encoder reconstructed buffers.

Field	Data Type	Input/ Output	Description
			Not required for encoder input buffer
contentType	XDAS_Int32	Input/Output	Video content type. See IVIDEO_ContentType enumeration in Table 4-1 for more details. This is useful when the content is both interlaced and progressive. The display process can use this field to determine how to render the display buffer.
chromaFormat	XDAS_Int32	Input/Output	Chroma format for encoder input data/decoded output buffer. See XDM_ChromaFormat enumeration in Table 4-1 for more details.
scalingWidth	XDAS_Int32	Input/Output	Scaled image width for post processing for decoder. Not applicable for encoders.
scalingHeight	XDAS_Int32	Input/Output	Scaled image height for post processing for decoder. Not applicable for encoders.
rangeMappingLuma	XDAS_Int32	Input/Output	Applicable for VC1, set to -1 as default for other codecs
rangeMappingChroma	XDAS_Int32	Input/Output	Applicable for VC1, set to -1 as default for other codecs
enableRangeReductionFlag	XDAS_Int32	Input/Output	Flag indicating whether to enable range reduction or not. Valid values are XDAS_TRUE and XDAS_FALSE. Applicable only for VC-1

Figure 4-3 shows `IVIDEO2_BufDesc` structure with the associated variables.

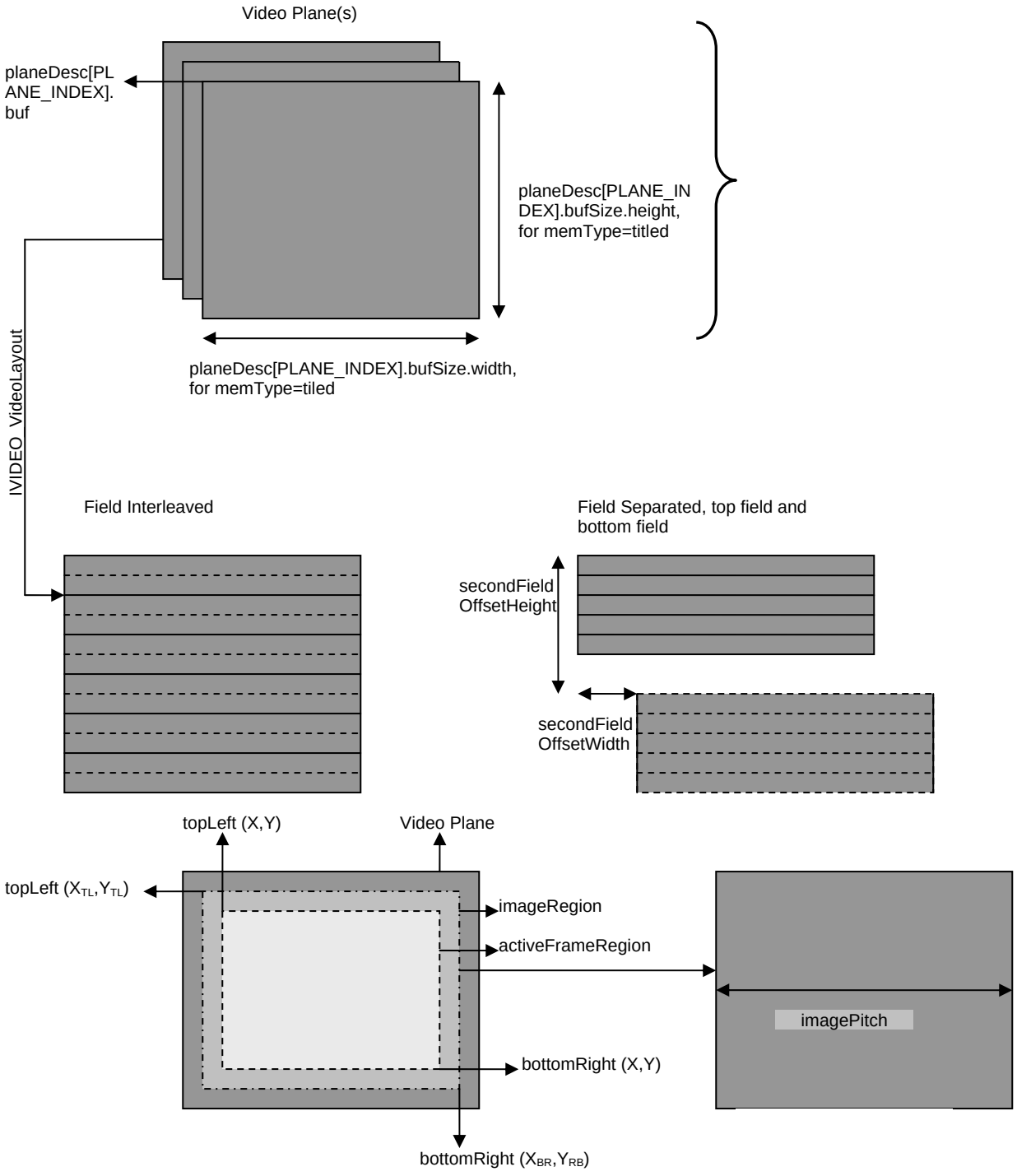


Figure 4-3. IVIDEO2_BufDesc With Associated Parameters.

4.2.1.6 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. See section 4.4 for more information
*control	XDAS_Int32	Input	Pointer to the control() function. See section 4.4 for more information

4.2.1.7 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. For the default and supported values, see Table 4-6.

Fields

Field	Data Type	Input/ Output	Description
Size	XDAS_Int32	Input	Size of the base or extended (if being used) data structure in bytes. Supported Values: • sizeof(IVIDENC2_Params) • sizeof(IMPEG4ENC_Params)
encodingPreset	XDAS_Int32	Input	Preset to control encoder quality. See XDM_EncodingPreset enumeration in Table 4-1 for more details.
rateControlPreset	XDAS_Int32	Input	Preset to control rate control selection. See IVIDEO_RateControlPreset enumeration in Table 4-1 for more details.
maxHeight	XDAS_Int32	Input	Maximum video height to be supported in pixels. Maximum height should be multiple of 2.

Field	Data Type	Input/ Output	Description
maxWidth	XDAS_Int32	Input	Maximum video width to be supported in pixels. Maximum width should be multiple of 16.
maxBitRate	XDAS_Int32	Input	Maximum bit-rate to be supported in bits per second. This field is ignored by the current version of the encoder.
minBitRate	XDAS_Int32	Input	Minimum Bit Rate for encoding in bits per second. This field is ignored by the current version of the encoder.
dataEndianness	XDAS_Int32	Input	Endianness of output data. See XDM_DataFormat enumeration in Table 4-1for more details.
maxInterFrameInterval	XDAS_Int32	Input	This is used for setting the maximum number of B frames between two reference frames. Distance from I-frame to P-frame: Currently B frames are not supported in MPEG SP encoder, hence this field is ignored by the encoder.
inputChromaFormat	XDAS_Int32	Input	Chroma format for the input buffer. See XDM_ChromaFormat enumeration in Table 4-1 for more details.
inputContentType	XDAS_Int32	Input	Video content type of the buffer being encoded. See IVIDEO_ContentType enumeration in Table 4-1 for more details. In this version of encoder, only IVIDEO_PROGRESSIVE is supported.
operatingMode	XDAS_Int32	Input	Video coding mode of operation.. See IVIDEO_OperatingMode enumeration in Table 4-1 for details
Profile	XDAS_Int32	Input	Profile indicator of video encoder. Only simple and baseline profile is supported in current version of the MPEG4 and H263 Encoder respectively.
Level	XDAS_Int32	Input	Level Indicator of video encoder. See IMPEG4VENC_Level enumeration in Table 4-2 for details.

Field	Data Type	Input/ Output	Description
inputDataMode	XDAS_Int32	Input	Input data mode. See IVIDEO_DataMode enumeration in Table 4-1for details. See in Appendix C for more details on datasynch usage with MPEG4 Encoder.
outputDataMode	XDAS_Int32	Input	Output data mode. See IVIDEO_DataMode enumeration in Table 4-1 for details. See in Appendix C for more details on datasynch usage with MPEG4 Encoder.
numInputDataUnits	XDAS_Int32	Input	Number of input slices/rows. Units depend on the inputDataMode, such as number of slices/rows/blocks, and so on. Ignored if inputDataMode is set to full frame mode.
numOutputDataUnits	XDAS_Int32	Input	Number of output slices/rows. Units depend on the outputDataMode, such as number of slices/rows/blocks, and so on. Ignored if outputDataMode is set to full frame mode.

Note:

For the supported maxBitRate values, see Table A.1 – Level Limits in ISO/IEC 14496-2:2003.

The following fields of IVIDENC2_Params data structure are level dependent:

- maxHeight
- maxWidth

To check the values supported for maxHeight and maxWidth use the following expression:

maxFrameSizeinMbs >= (maxHeight*maxWidth) / 256;

See Table A.1 – Level Limits in ISO/IEC 14496-2:2003 for the supported maxFrameSizeinMbs values.

For example, consider you have to check if the following values are supported for level L2:

- maxHeight = 480
- maxWidth = 720

The supported maxFrameSizeinMbs value for level 2.0 as per Table N.1 – Level Limits is 396.

Compute the expression as:

```
maxFrameSizeinMbs >= (480*720) / 256
```

The value of `maxFrameSizeinmbs` is 1350 and hence the condition is not true. Therefore, the above values of `maxHeight` and `maxWidth` are not supported for level 2.0.

See Table N.1 – Level Limits in ISO/IEC 14496-2:2003 for the supported values of `maxMbsPerSecond`.

Use the following expression to calculate `FrameSizeinMbs`:

```
FrameSizeinMbs = (maxWidth * maxHeight) / 256;
```

4.2.1.8 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. For the default and supported values, see Table 4-7.

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	Height of input frame in pixels. Input Height needs to be multiple of 2.
`inputWidth`	`XDAS_Int32`	Input	Width of input frame in pixels. Input width can be changed before start of encoding with in the limits of Max width set in creation phase. `inputWidth` must be multiples of two. Minimum width supported is 64. **Note:** • For MPEG4, width can be multiple of 2 but for H263 with custom picture format, width should be multiple of 4. • When the input width is a non-multiple of 16, the encoder expects the application to pad the input frame to the nearest multiple of 16 to the right of the frame. In this case, application shall set `inputWidth` to actual width but should provide the padded input YUV data buffer to encoder.
`refFrameRate`	`XDAS_Int32`	Input	Reference or input frame rate in fps * 1000. For example, if the frame rate is 30, set this field to 30000.
`targetFrameRate`	`XDAS_Int32`	Input	Target frame rate in fps * 1000. For example, if the frame rate is 30, set this field to 30000. Note that both refFrameRate and targetFrameRate should have the same value for the encoder input.

Field	Data Type	Input/ Output	Description
targetBitRate	XDAS_Int32	Input	Target bit-rate in bits per second. For example, if the bit-rate is 2 Mbps, set this field to 2097152.
intraFrameInterval	XDAS_Int32	Input	Interval between two consecutive intra frames. For example: • 0 – One I frame followed by all Inter Frames • 1 - No inter frames (all intra frames) • N - One intra frame and N-1 inter frames, where N > 0.
generateHeader	XDAS_Int32	Input	Encode entire access unit or only header. See XDM_EncMode enumeration for details.
captureWidth	XDAS_Int32	Input	If the field is set to: • 0 - Encoded image width is used as pitch. • Any non-zero value, capture width is used as pitch (if capture width is greater than image width). This value should be multiple of 16.
forceFrame	XDAS_Int32	Input	Force the current (immediate) frame to be encoded as a specific frame type.
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. DEFAULT(0). Currently not supported in this version of MPEG4 Encoder, hence this value needs to be zero.
mvAccuracy	XDAS_Int32	Input	Pixel accuracy of the motion vector. See IVIDENC2_MotionVectorAccuracy enumeration in Table 4-1 for details. Only Half pel accuracy is supported in this version of MPEG4 encoder.
sampleAspectRatioHeight	XDAS_Int32	Input	Sample aspect ratio height. This will be considered by encoder only when IVIDENC2_DynamicParams :: aspectRatioIdc is IMPEG4ENC_ASPECTRATIO_EXTENDED
sampleAspectRatioWidth	XDAS_Int32	Input	Sample aspect ratio width. This will be considered by encoder only when IVIDENC2_DynamicParams :: aspectRatioIdc is IMPEG4ENC_ASPECTRATIO_EXTENDED
ignoreOutbufSizeFlag	XDAS_Int32	Input	Flag to indicate that for bit-stream buffer size, codec to expect the requested size or not Valid values are XDAS_TRUE and XDAS_FALSE.

Field	Data Type	Input/ Output	Description
putDataFxn	XDM_DataSyncPutFxn	Input	Function pointer to produce data at sub-frame level
putDataHandle	XDM_DataSyncHandle	Input	Handle that identifies the data sync FIFO and is passed as argument to putData calls
getDataFxn	XDM_DataSyncPutFxn	Input	Function pointer to receive data at sub-frame level
getDataHandle	XDM_DataSyncHandle	Input	Handle that identifies the data sync FIFO and is passed as argument to getData calls
getBufferFxn	XDM_DataSyncPutFxn	Input	Function to receive buffer at sub frame level
getBufferHandle	XDM_DataSyncHandle	Input	Handle that identifies the datasync FIFO and is passed as argument to getBufferFxn calls
lateAcquireArg	XDAS_Int32	Input	Argument used during late acquire, For all control() commands other than #XDM_SETLATEACQUIREARG, this field is ignored and can therefore be set by the caller to any value. This is a identifier for a channel in multi channel scenario.

4.2.1.9 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. Zero (0) is not a supported inputID. This value is reserved for cases when there no input buffer is provided. This is useful when frames require buffering (example, B frames) and to support buffer management. When there is no re-ordering, IVIDENC2_OutArgs:: freeBufId will be the same as this inputID field.
control	XDAS_Int32	Input	Encoder control operations. By this parameter various control operations like forcing a frame to be SKIP can

Field	Data Type	Input/ Output	Description
			be achieved, See IVIDENC2_Control enumeration for more details.

4.2.1.10 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 in Table 4-1 for details.
Data	XDM1_SingleBufDesc	Output	Buffer descriptor for data passing If this field is not used, the application must set data.buf to NULL. This buffer can be used as either input or output, depending on the command. The buffer will be provided by the application, and returned to the application on return of the IVIDENC1_Fxns.control() call. The algorithm must not retain a pointer to this data.
encodingPreset	XDAS_Int32	Output	Encoding preset. See XDM_EncodingPreset enumeration in Table 4-1 for details.
rateControlPreset	XDAS_Int32	Output	Rate control preset See IVIDEO_RateControlPreset enumeration in Table 4-1 for details.
maxInterFrameInterval	XDAS_Int32	Output	This is used for setting the maximum number of B frames between two reference frames. Distance from I-frame to P-frame: B frames are not supported in the current MPEG4 Encoder.
inputChromaFormat	XDAS_Int32	Output	Chroma format for the input buffer. See XDM_ChromaFormat enumeration in Table 4-1for details.

Field	Data Type	Input/ Output	Description
inputContentType	XDAS_Int32	Output	Video content type of the buffer being encoded. See IVIDEO_ContentType enumeration in Table 4-1 for details.
operatingMode	XDAS_Int32	Output	Mode of video coding. See IVIDEO_OperatingMode enumeration in Table 4-1for details
profile	XDAS_Int32	Output	Profile indicator of video encoder.
level	XDAS_Int32	Output	Level indicator of video encoder. See IMEPG4VENC_Level enumeration in Table 4-2 for details.
inputDataMode	XDAS_Int32	Output	Input data mode. See IVIDEO_DataMode enumeration n Table 4-1for details.
outputDataMode	XDAS_Int32	Output	Output data Mode. See IVIDEO_DataMode enumeration n Table 4-1 for details.
bufInfo	XDM1_AlgBufInf	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. See XDM1_AlgBufInfo data structure for details.
encDynamicParams	IVIDENC2_DynamicParams	Output	Dynamic parameters in use by encoder. See IVIDENC2_DynamicParams enumeration for more details. In case of extended dynamic parameters, algorithm can check the size of Status or DynamicParams and return the parameters accordingly.

4.2.1.11 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	Extended error code. See XDM_ErrorBit enumeration in Table 4-1for details.
bytesGenerated	XDAS_Int32	Output	The number of bytes generated during the IVIDENC2_Fxns::process() call.
encodedFrameType	XDAS_Int32	Output	Frame types for video. See IVIDEO_FrameType enumeration in Table 4-1for details.
inputFrameSkip	XDAS_Int32	Output	Frame skipping modes for video. See IVIDEO_SkipMode enumeration in Table 4-1for details.
freeBufID[IVIDEO2_MAX_IO_BUFFERS]	XDAS_Int32	Output	This is an array of input IDs corresponding to the buffers that have been unlocked in the current process call. The first zero entry in array will indicate end of valid freeBufIDs within the array Buffers returned to the application for display (through IVIDDEC2_OutArgs#displayBufs) continue to be owned by the algorithm until they are released - indicated by the ID being returned in this freeBuf array. The buffers released by the algorithm are indicated by their non-zero ID (previously provided through IVIDDEC2_InArgs#inputID). A value of zero (0) indicates an invalid ID. The first zero entry in array will indicate end of valid freeBufIDs within the array. Hence, the application can stop searching the array when it encounters the first zero entry. If no buffer was unlocked in the process call, freeBufID[0] will have a value of zero.
reconBufs	IVIDEO2_BufDesc	Output	Pointer to re-construction buffer descriptor. See IVIDEO2_BufDesc data structure for more information

4.2.1.12 XDM_Date

Description

This structure contains the date and time information.

Fields

Field	Data Type	Input/ Output	Description
msecsOfDay	XDAS_Int32	Input	Milliseconds of the day
month	XDAS_Int32	Input	Month (0 = January, 11 = December)
dayOfMonth	XDAS_Int32	Input	Day (1 - 31)
dayOfWeek	XDAS_Int32	Input	Day of week (0 = Sunday, 6 = Saturday)
year	XDAS_Int32	Input	Year (since 0)

4.2.1.13 XDM_Point

Description

This structure specifies the two dimensional point.

Fields

Field	Data Type	Input/ Output	Description
X	XDAS_Int32	Input	X field of the frame
Y	XDAS_Int32	Input	Y field of the frame

4.2.1.14 XDM_Rect

Description

This structure defines the region in the image that is to be encoded.

Fields

Field	Data Type	Input/ Output	Description
topLeft	XDM_Point	Input	Top left corner of the frame. See XDM_Point data structure for details.
bottomRight	XDM_Point	Input	Bottom right corner of the frame. See XDM_Point data structure for details.

4.2.1.15 XDM_DataSyncDesc

Description

This structure provides the descriptor for the chunk of data being transferred in one call to putData or getData function.

This DataSyncDesc Syntax is common to Input Data Synchronization and Output Data Synchronization implementation.

Fields

Field	Data Type	Input/ Output	Description
size	XDAS_Int32	Input/ Output	Size of this structure
*baseAddr	XDAS_Int32	Input/ Output	Input: not used Output: Address of first byte on first block
numBlocks	XDAS_Int32	Input/ Output	Input: Number of input rows available in the frame buffer to encode Output: Number of 2 KBytes buffers available to put the encoded bit stream data
varBlockSizesFlag	XDAS_Int32	Input/ Output	Flag indicating whether any of the data blocks vary in size. Valid values are XDAS_TRUE and XDAS_FALSE. varBlockSizesFlag with XDAS_TRUE is not supported.
*blockSizes	XDAS_Int32	Input/ Output	Variable block sizes array. If varBlockSizesFlag is XDAS_TRUE, this array contains the sizes of each block. If varBlockSizesFlag is XDAS_FALSE, this array contains the size of same-size blocks. Memory for this array (of size numBlocks) has to be allocated by the caller of the putData or getdata API.

4.2.2 MPEG4 Encoder Data Structures

This section includes the following MPEG4 Encoder specific extended data structures:

- IMPEG4ENC_Params
- IMPEG4ENC_RateControlParams
- IMPEG4ENC_InterCodingParams
- IMPEG4ENC_IntraCodingParams
- IMPEG4ENC_SliceCodingParams
- IMPEG4ENC_DynamicParams
- IMPEG4ENC_Inargs
- IMPEG4ENC_Status
- IMPEG4ENC_OutArgs
- IMPEG4ENC_Fxns
- IMPEG4ENC_DataSyncDesc

4.2.2.1 IMPEG4ENC_Params

Description

This structure defines the creation parameters and any other implementation specific parameters for a MPEG4 Encoder instance object. The creation parameters are defined in the XDM data structure, IVIDENC2_Params. For the default and supported values, see Table 4-.

Fields

Field	Data Type	Input/ Output	Description
videnc2Params	IVIDENC2_Params	Input	See IVIDENC2_Params data structure for details.
rateControlParams	IMPEG4ENC_RateControlParams	Input	Controls all rate control related parameters. See IMPEG4ENC_RateControlParams data structure for details.
interCodingParams	IMPEG4ENC_InterCodingParams	Input	Controls all inter coding related parameters. See IMPEG4ENC_InterCodingParams data structure for details.
intraCodingParams	IMPEG4ENC_IntraCodingParams	Input	Controls all intra coding related parameters. See IMPEG4ENC_IntraCodingParams data structure for details.
sliceCodingParams	IMPEG4ENC_SliceCodingParams	Input	Controls all Slice coding related parameters. See IMPEG4ENC_SliceCodingParams data structure for details.

Field	Data Type	Input/ Output	Description
useDataPartitioning	XDAS_UInt32	Input	Data partition mode on/off. • 1 - ON • 0 - OFF Default value is 0. This mode can be switched ON only when sliceMode is IMPEG4_SLICEMODE_BITS and non-zero sliceUnitSize or else MPEG4 Encoder returns error. If useShortVideoHeader is enabled, MPEG4 Encoder automatically disables this mode. Also note that, with DP ON, • bitrate and packet adherence is not guaranteed. • Does not support output data synch mode. Codec returns error with this combination.
useRvlc	XDAS_UInt32	Input	RVLC mode ON/OFF. • 1 - ON • 0 - OFF Default value is 0. This mode can be switched on only when useDataPartitioning is ON or else MPEG4 Encoder return error. If useShortVideoHeader is enabled, MPEG4 Encoder automatically disable this mode.
useShortVideoHeader	XDAS_UInt32	Input	Controls the encoding format. 1 - H.263 0 - MPEG4 Default value is 0. If useShortVideoHeader is 1, MPEG4 Encoder automatically sets the minBlockSizeP to IMPEG4_BLOCKSIZE_16x16 . But can have H.241 (resyncIntervalInbits) feature with outputMode set to SLICE_MODE. See more details about this in Appendix A.
vopTimeIncrementResolution	XDAS_UInt32	Input	Resolution of vop_time_increment bit-stream syntax element, number of ticks/sec. Default value is 30.

Field	Data Type	Input/ Output	Description
nonMultiple16RefPadMethod	XDAS_UInt32	Input	Controls the way the padding is done for Ref Frame when Height is Non-multiple of 16. • IMPEG4_PAD_METHOD_MPEG4 – MPEG4 Standard specific way of padding • IMPEG4_PAD_METHOD_DIVX – VLC, DIVx way of padding Default value is IMPEG4_PAD_METHOD_MPEG4.
pixelRange	XDAS_UInt32	Input	Pixel range to be put in header See IMPEG4VENC_PixelRange enumeration for details. Default value is IMPEG4VENC_PR_DEFAULT.
enableSceneChangeAlgo	XDAS_UInt32	Input	This parameter enables or disables the scene change detection algorithm to be used by the codec. IMPEG4ENC_SCDA_ENABLE -> Enable the scene change detection algorithm. IMPEG4ENC_SCDA_DISABLE -> Disable the scene change detection algorithm. **Note** When Scene Change Detection algo is enabled, based on the algorithm logic, Intra frames can be inserted forcefully during the scene change in the video sequence. Usually not recommended for CBR (low delay) rate control since it is tough to meet the bitrate if frequent I frames get inserted. Also not recommended in H241 slice mechanism as I frame generates more slices than P frames at the cost of performance. Also note that SCD algo is disabled by the codec automatically whenever early frame skip is enabled i.e. allowFrameSkip is 2.
useVOS	XDAS_UInt32	Input	This parameter enables or disables the insertion of video object sequence header in the bitstream. 1 -> Enable inserting VOS header 0 -> Disable inserting VOS header

Field	Data Type	Input/ Output	Description
enableMONA	XDAS_UInt32	Input	Flag to indicate that the encoder should enable MONA mode settings and override any previous MPEG4 encoder initializations. It should also neglect any encoder initializations after this flag is enabled. 1 - Enable MONA settings 0 – Disable MONA settings and use normal encoder settings **Note:** **MONA Settings for MPEG-4:** targetBitRate <= 64000 levelIdc = Level 0 refFrameRate <= 15fps targetFrameRate <= 15fps aspectRatio = 1 timerResolution = 30000 inputWidth = 176 inputHeight = 144 packetSize <= 2048 useDataPartition = 1 useRVLC = 0 rcAlgo = IMP4VENC_RC_CBR qpMax = 31 qpMin = 1 **MONA Settings for H.263:** targetBitRate <= 64000 refFrameRate <= 15fps targetFrameRate <=15fps timerResolution = 30000 inputWidth = 176 inputHeight = 144 rcAlgo = IMP4VENC_RC_CBR qpMax = 31 qpMin = 8
enableAnalyticinfo	XDAS_UInt32	Input	Flag to enable or disable the dumping of Motion vector and SAD info from the codec to the application. 1 -> Enable dumping of MV info and SAD 0 -> Disable dumping of MV info and SAD See in Appendix D for more details on the implementation and interface details.
debugTraceLevel	XDAS_UInt32	Input	This parameter configures the codec to dump a debug trace log. 0-> Disables the dumping of debug trace parameter. 1 -> Frame level information Note that this parameter is only used for debugging.

Field	Data Type	Input/ Output	Description
lastNFramesToLog	XDAS_UInt32	Input	This parameters 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. This parameter is valid if debugTraceLevel is enabled.

4.2.2.2 IMPEG4ENC_RateControlParams

Description

This structure controls rate control behavior. For the default and supported values, see Table 4-8.

Fields

Field	Data Type	Input/ Output	Description
rateControlParamsPreset	IMPEG4ENC_RateControlParamsPreset	Input	This preset controls the USER_DEFINED versus DEFAULT mode. If user are not aware of the below fields, it should be set as IMPEG4_RATECONTROLPARAMS_DEFAULT
rcAlgo	IMPEG4ENC_RateControlAlgo	Input	This defines the rate control algorithm to be used. • 0 - IMPEG4_RATECONTROLALGO_NONE • 1 - IMPEG4_RATECONTROLALGO_VBR • 2 - IMPEG4_RATECONTROLALGO_CBR default value is IMPEG4_RATECONTROLALGO_VBR .
qpI	XDAS_Int32	Input	Initial quantization parameter for I frames. Valid range is 1 to 31. When rateControlPreset = IVIDEO_NONE, this quantization parameter is used by the video frame/field. default value is 5.
qpP	XDAS_Int32	Input	Initial quantization parameter for P frames. Valid Range is 1 to 31. When rateControlPreset = IVIDEO_NONE, this quantization parameter is used by the whole video frame/field. default value is 5.
seIntialQP	XDAS_Int32	Input	Initial quantization parameter for the first frame in the sequence when RC is ON. Valid range is 1 to 31.

Field	Data Type	Input/ Output	Description
			When rcAlgo is not IMPEG4_RATECONTROLALGO_NONE this value is used as the initial QP when Rate Control is on. Default value is 5. If user is not aware of what this value needs to be, then specify this value as ZERO, where by codec would internally calculate optimal initial QP.
qpMax	XDAS_Int32	Input	Maximum quantization parameter for I/P frame(s). Range is 1 to 31. Default Value is 31.
qpMin	XDAS_Int32	Input	Minimum quantization parameter for I/P frame(s). Range is 1 to 31. Default Value is 1. Note: qpMin shall not be greater than qpMax or else MPEG4 Encoder return error.
enablePerceptualQuantMode	XDAS_Int32	Input	Controls the MB Level QP change in a row by Perceptual Quantization Method. • 1 – Enabled • 0 - Disabled Default value is 0. MPEG4 Encoder automatically disables enablePerceptualQuantMode when low Delay RC (CBR) is enabled or H.241 or DP/RVLC modes enabled.
allowFrameSkip	XDAS_Int32	Input	Controls the frame skip feature based on the bit consumption. • 2 – Enabling early frame skip • 1 – Enabling late frame skip • 0 – Disabling frame skip Default value is 0. MPEG4 Encoder automatically enable this mode when low Delay RC(CBR) is enabled. **Note**: When user selects output data mode as IVIDEO_FIXEDLENGTH or IVIDEO_SLICEMODE then codec assumes the value of allowFrameSkip as 2. That is early frame skip will be enabled whenever output data synch is enabled.
initialBufferLevel	XDAS_Int32	Input	Initial buffer level for VBV compliance. It informs that hypothetical decoder can start decoding on the fullness of the VBV buffer. See the Table 4-4 for the range of value supported by this

Field	Data Type	Input/ Output	Description
			parameter. Default value is 0, where codec will internally calculate the value based on the RC algo type.
vbvBufferSize	XDAS_Int32	Input	Virtual Buffer Verifier buffer size. This size controls the frame skip logic of the encoder. For low delay applications this size should be small. This size is in bits. Maximum Value is level dependant and min value is `0.1* TargetBitRate`. Default value is 0, where codec will internally calculate the value based on the RC algo type. See the Table 4-5 for the range of value supported by this parameter.
qpMinIntra	XDAS_Int32	Input	Minimum quantization parameter for Intra macroblocks in Inter frames. Codec will not always encode all the Intra macroblocks in Inter frames with this value but tries to encode with this user configured QP value. This depends on the QP of the previous macroblock of the Intra macroblock. As per H.263, standard, adjacent QP values can vary between +/-2. So if the previous macroblock has QP as qpPrev, then the Intra macroblock QP will be decided as follows: min(qpPrev + 2, qpMinIntra) This variable is applicable only to H.263 and not applicable to MPEG-4. This is only applicable to Intra macroblocks in Inter frames and not applicable to Intra macroblocks in Intra frame. Default value is 0 Valid values: 0, 1-31 0 – No effect in QP values for Intra macroblocks in Inter frame. Used for backward compatibility.

Note:

The following parameters are considered during run-time:

- `qpI`
- `qpMax`
- `qpMin`
- `qpP`
- `qpMinIntra`
- `allowFrameSkip` for VBR (storage). For CBR, codec always assumes allowFrameSkip is enabled.

The following parameters are ignored during run-time:

- rcAlgo
- initialBufferLevel
- vbvBufferSize
- enablePerceptualQuantMode
- seInitialQP

4.2.2.3 IMPEG4ENC_InterCodingParams

Description

This structure contains all the parameters which controls inter MBs coding behavior. For the default and supported values, see Table 4-9.

Fields

Field	Data Type	Input/ Output	Description
interCodingPreset	IMPEG4ENC_InterCodingPreset	Input	This preset controls the USER_DEFINED versus DEFAULT mode. If user is not aware of below fields, it should be set as IMPEG4_INTERCODING_DEFAULT
searchRangeHorP	XDAS_Int32	Input	Horizontal search range for P frames Possible values: Non zero, maximum up to 144.
searchRangeVerP	XDAS_Int32	Input	Vertical search range for P frames Possible Values: Non-zero, maximum up to 32
globalOffsetME	XDAS_Int32	Input	ME with global offset. • 0 – Disable • 1 – Enable Default value is 1.
earlySkipThreshold	XDAS_Int32	Input	Threshold to use for early skip decision of a macroblock. Default value is 200.
enableThresholdingMethod	XDAS_Int32	Input	Threshold cost method used to set a block to be not_coded if the block has very few small amplitude coefficients,.Can take values 0 and 1. Default value is 1.
minBlockSizeP	XDAS_Int32	Input	Minimum Block size for MB coding in P Frame. • 0 – IMPEG4_BLOCKSIZE_16x16 -- 16x16 block size • 1 – IMPEG4_BLOCKSIZE_8x8 -- 8x8 block size Default value is IMPEG4_BLOCKSIZE_8x8.
enableRoundingControl	XDAS_Int32	Input	When enabled reduces the IDCT drift.

Field	Data Type	Input/ Output	Description
			• 0 – Disable • 1 – Enable Default value is 1.

4.2.2.4 IMPEG4ENC_IntraCodingParams

Description

This structure defines all the operations on MPEG4 Encoder instance objects. For the default and supported values, see Table 4-10.

Fields

Field	Data Type	Input/ Output	Description
intraCodingPreset	IMPEG4ENC_IntraCodingPreset	Input	This preset controls the USER_DEFINED versus DEFAULT mode. If user is not aware of below fields, it should be set as IMPEG4_INTRACODING_DEFAULT
intraRefreshMethod	XDAS_UInt32	Input	Specifies the Adaptive Intra Refresh method to be used • 0 – IMPEG4_INTRAREFRESH_NONE • 1 – IMPEG4_INTRAREFRESH_CYCLIC_MBS • 2 – IMPEG4_INTRAREFRESH_CYCLIC_ROWS • 3 –IMPEG4_INTRAREFRESH_MANDATORY • 4 – IMPEG4_INTRAREFRESH_RDOPT_MBS Default value is 0. Current version of MPEG4 Encoder does not support value 4.
intraRefreshRate	XDAS_UInt32	Input	Specifies the additional parameter for the intraRefreshMethod. When intraRefreshMethod== IMPEG4_INTRAREFRESH_CYCLIC_MBS, this field provides the information of number of MBs after which a MB need to be coded as intra. When intraRefreshMethod== IMPEG4_INTRAREFRESH_CYCLIC_ROWS, this field provides the information of number of Rows to be refreshed as Intra per frame. Default value is 0.
acpredEnable	XDAS_UInt32	Input	Flag to switch ON/OFF ac prediction for I frames/MBs • 0 – Disable ac prediction

Field	Data Type	Input/ Output	Description
			• 1 – Enable ac prediction Default value is 1.
insertGOVHdrBeforeIframe	XDAS_UInt32	Input	If non-zero inserts GOV Header before I Frame. Default value is 0.
enableDriftControl	XDAS_UInt32	Input	If non-zero, enables drift control. Default value is 1.

4.2.2.5 IMPEG4ENC_SliceCodingParams

Description

This structure contains all the parameters which controls slice encoding. For the default and supported values, see Table 4-11.

Fields

Field	Data Type	Input/ Output	Description
sliceCodingPreset	IMPEG4ENC_SliceCodingPreset	Input	This preset controls the USER_DEFINED versus DEFAULT mode. If user is not aware of below fields, it should be set as IMPEG4_SLICECODING_DEFAULT
sliceMode	XDAS_UInt32	Input	Can take values of • IMPEG4_SLICEMODE_NONE – No slices are generated • IMPEG4_SLICEMODE_MBUNIT –slices are generated based on maximum number of MBs provided by sliceUnitSize. This is not supported with short video header mode. • IMPEG4_SLICEMODE_BITS – slices are generated based on the bits consumptions whose max is provided by sliceUnitSize; Also used to support H.241 Default value is IMPEG4_SLICEMODE_NONE
sliceUnitSize	XDAS_UInt32	Input	When sliceMode is IMPEG4_SLICEMODE_MBUNIT, sliceUnitSize assumes the definition of maximum MBs Per Slice. Unit is in number of Macroblocks. When sliceMode is IMPEG4_SLICEMODE_BITS, sliceUnitSize assumes the definition of maximum Bits Per Slice. Unit is in number of bits. If IMPEG4_SLICEMODE_BITS is non zero, MPEG4 Encoder automatically disable the enablePerceptualQuantMode.
gobInterval	XDAS_UInt32	Input	Insert GOB header after every n GOBs in H263 encoding mode. Valid values: 0 to maximum number of rows in a frame.
useHec	XDAS_UInt32	Input	Header Extension Code on/off. Only applicable incase of MPEG4 but not in H263 • 0 – No Header Extension Code

Field	Data Type	Input/ Output	Description
			• 1 – Use HEC only after first RM • 2 – Use HEC after all RM's except the first RM Default value is 0.

4.2.2.6 IMPEG4ENC_DynamicParams

Description

This structure defines the run-time parameters and any other implementation specific parameters for a MPEG4 Encoder instance object. The run-time parameters are defined in the XDM data structure, `IVIDENC2_DynamicParams`. For the default and supported values, see Table 4-12.

Fields

Field	Data Type	Input/ Output	Description
`videnc2DynamicParams`	`IVIDENC2_DynamicParams`	Input	See `IVIDENC2_DynamicParams` data structure for details.
`rateControlParams`	`IMPEG4ENC_RateControlParams`	Input	Controls all rate control related parameters. Only few are supported to be changed as part control call. See `IMPEG4ENC_RateControlParams` data structure for more details.
`interCodingParams`	`IMPEG4ENC_InterCodingParams`	Input	Controls all inter MB coding related parameters. Only few are supported to be changed as part control call. See `IMPEG4ENC_InterCodingParams` data structure for more details
`sliceCodingParams`	`IMPEG4ENC_SliceCodingParams`	Input	Controls all slice coding related parameters. See `IMPEG4ENC_SliceCodingParams` data structure for more details.

Field	Data Type	Input/ Output	Description
aspectRatioIdc	XDAS_UInt32	Input	Encoder inserts aspectRatioIdc as it is in the bit-stream. It is user's responsibility to input appropriate value. See Table 6-14 of MPEG4 standard or enum IMPEG4ENC_AspectRatioIdc for valid values. When aspectRatioIdc == IMPEG4ENC_ASPECTRATIO_EXTENDED(15), encoder will look at IVIDENC2_DynamicParams::sampleAspectRatioHeight and IVIDENC2_DynamicParams::sampleAspectRatioWidth and use them as sar_height and sar_width respectively. aspectRatioIdc is left to user to provide correct value. For extended PAR, the sampleAspectRatioHeight and sampleAspectRatioWidth should be relatively prime as per the standard. if aspectRatioIdc ==0 then encoder uses 1:1 as default picture aspect ratio. **Note** that aspectRatioIdc is valid for MPEG4 and H263 custom picture formats.

Note:

Any field from the IMPEG4VENC_DynamicParams structure is useful only when the encodingPreset field of IVIDENC2_Params data structure is equal to XDM_USER_DEFINED.

4.2.2.7 IMPEG4ENC_Inargs

‖ Description

This structure defines the run-time input arguments for MPEG4 SP Encoder instance object.

‖ Fields

Field	Data Type	Input/ Output	Description
videnc2InArgs	IVIDENC2_Inargs	Input	See IVIDENC2_Inargs data structure for details

4.2.2.8 IMPEG4ENC_Status

‖ Description

This structure defines parameters that describe the status of the MPEG4 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
`Videnc2Status`	`IVIDENC2_Status`	Output	See `IVIDENC2_Status` data structure for details. Status of the MPEG4 encoder along with error information, if any.
`rateControlParams`	`IMPEG4ENC_RateControlParams`	Output	See `IMPEG4ENC_RateControlParams` data structure for details.
`interCodingParams`	`IMPEG4ENC_InterCodingParams`	Output	See `IMPEG4ENC_InterCodingParams` data structure for details.
`intraCodingParams`	`IMPEG4ENC_IntraCodingParams`	Output	See `IMPEG4ENC_IntraCodingParams` data structure for details.
`sliceCodingParams`	`IMPEG4ENC_SliceCodingParams`	Output	See `IMPEG4ENC_SliceCodingParams` data structure for details.
`useDataPartitioning`	`XDAS_UInt32`	Output	Controls data partitioning for MPEG4 Encoder.
`useRvlc`	`XDAS_UInt32`	Output	Controls RVLC for MPEG4 Encoder.
`useShortVideoHeader`	`XDAS_UInt32`	Output	Controls the encoding mode i.e. MPEG4 or H263.
`vopTimeIncrementResolution`	`XDAS_UInt32`	Output	Resolution of `vop_time_increment` bit-stream syntax element, number of ticks/sec.
`nonMultiple16RefPadMethod`	`XDAS_UInt32`	Output	Controls the way the padding is done for Ref Frame when Height is Non-multiple of 16. • `IMPEG4_PAD_METHOD_MPEG4` – MPEG4 Standard specific way of padding • `IMPEG4_PAD_METHOD_DIVX` – VLC, DIVx way of padding Default value is `IMPEG4_PAD_METHOD_MPEG4`.
`pixelRange`	`XDAS_UInt32`	Output	Pixel range to be put in header See `IMPEG4VENC_PixelRange` enumeration for details. Default value is IMPEG4VENC _PR_DEFAULT.

Field	Data Type	Input/ Output	Description
enableSceneChangeAlgo	XDAS_UInt32	Output	This parameter enables or disables the scene change detection algorithm to be used by the codec. IMPEG4ENC_SCDA_ENABLE -> Enable the scene change detection algorithm. Note that in this case, based on the algorithm logic, Intra frames can be inserted forcefully. IMPEG4ENC_SCDA_DISABLE -> Disable the scene change detection algorithm.
useVOS	XDAS_UInt32	Output	Control to enable or disable putting VOS header in the stream.
enableMONA	XDAS_UInt32	Output	Control to enable or disable MONA setting based encoding.
enableAnalyticinfo	XDAS_Int32	Output	Control to enable or disable dumping analytjc info by the codec.
debugTraceLevel	XDAS_UInt32	Output	Control to enable or disable to dump trace information.
lastNFramesToLog	XDAS_UInt32	Output	This parameters configures the codec to maintain history of debug trace parameters for last N frames
extMemoryDebugTraceAddr	XDAS_UInt32	Output	External memory address (as seen by Media Controller) where debug trace information is being dumped .
extMemoryDebugTraceSize	XDAS_UInt32	Output	External memory buffer size (in bytes) where debug trace information is being dumped - the size of last memory buffer

4.2.2.9 IMPEG4ENC_OutArgs

Description

This structure defines the run-time output parameters for the MPEG4 Encoder instance object.

Fields

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

4.2.2.10 IMPEG4ENC_Fxns

Description

This structure defines all the operations on MPEG4 SP Encoder instance objects.

Fields

Field	Data Type	Input/ Output	Description
ividenc	IVIDENC2_Fxns	Output	See IVIDENC2_Fxns data structure for details.

4.2.2.11 IMPEG4ENC_DataSyncDesc

Description

This structure is an extension of XDM_DataSyncDesc to provide additional information required for Mode B Packetization.

Fields

Field	Data Type	Input/ Output	Description
dataSyncDesc	XDM_DataSyncDesc	Output	See XDM_DataSyncDesc data structure for details.
mbAddr	XDAS_UInt16	Output	this is a pointer to an array of First MB Addresses in different slices
gobNumber	XDAS_UInt16	Output	this is a pointer to an array of GOB Number of first Mb in different slices
quantScale	XDAS_UInt16	Output	this is a pointer to an array of Quant values of first MB in different slices
mv	XDAS_UInt32	Output	this is a pointer to an array of MV of first MB in different slices. NOTE: Motion vectors are in Half-pel resolution

4.3 Default and Supported Values of Parameters

This section provides the default and supported values for the following data structures:

- IVIDENC2_Params
- IVIDENC2_DynamicParams
- IMPEG4ENC_RateControlParams
- IMPEG4ENC_InterCodingParams
- IMPEG4ENC_IntraCodingParams
- IMPEG4ENC_SliceCodingParams
- IMPEG4ENC_Params
- IMPEG4ENC_DynamicParams

Table 4-6. Default and Supported Values for IVIDENC2_Params

Field	Default Value	Supported Value
size	sizeof(IMPEG4ENC_Params)	• sizeof(IVIDENC2_Params) • sizeof(IMPEG4ENC_Params)
encodingPreset	XDM_DEFAULT	• XDM_DEFAULT • XDM_USER_DEFINED
rateControlPreset	IVIDEO_STORAGE	• IVIDEO_STORAGE • IVIDEO_LOWDELAY • IVIDEO_NONE • IVIDEO_USER_DEFINED
maxHeight	2048	[64, 2048]
maxWidth	2048	[64, 2048]
dataEndianness	XDM_BYTE	XDM_BYTE
maxInterFrameInterval	0	0
maxBitRate	24000000	Ignored. No error check.
minBitRate	0	Ignored.

Field	Default Value	Supported Value
		No error check.
inputContentType	IVIDEO_PROGRESSIVE	• IVIDEO_PROGRESSIVE
operatingMode	IVIDEO_ENCODE_ONLY	• IVIDEO_ENCODE_ONLY
profile	3 (Simple Profile)	3 (Simple Profile)
level	IMPEG4ENC_SP_LEVEL_5	• IMPEG4ENC_SP_LEVEL_0 • IMPEG4ENC_SP_LEVEL_0B • IMPEG4ENC_SP_LEVEL_1 • IMPEG4ENC_SP_LEVEL_2 • IMPEG4ENC_SP_LEVEL_3 • IMPEG4ENC_SP_LEVEL_4A • IMPEG4ENC_SP_LEVEL_5 • IMPEG4ENC_SP_LEVEL_6 • IMPEG4ENC_H263_LEVEL_10 • IMPEG4ENC_H263_LEVEL_20 • IMPEG4ENC_H263_LEVEL_30 • IMPEG4ENC_H263_LEVEL_40 • IMPEG4ENC_H263_LEVEL_45 • IMPEG4ENC_H263_LEVEL_50 • IMPEG4ENC_H263_LEVEL_60 • IMPEG4ENC_H263_LEVEL_70
inputDataMode	IVIDEO_ENTIREFRAME	• IVIDEO_ENTIREFRAME • IVIDEO_NUMROWS
outputDataMode	IVIDEO_ENTIREFRAME	• IVIDEO_ENTIREFRAME • IVIDEO_FIXEDLENGTH • IVIDEO_SLICEMODE
numInputDataUnits	1	• Any value > 0 and <= 8 when inputDataMode is IVIDEO_NUMROWS. Ignored incase of IVIDEO_ENTIREFRAME.

Field	Default Value	Supported Value
numOutputDataUnits	1	• Any value > 0 and <= 8 when outputDataMode is either IVIDEO_FIXEDLENGTH or IVIDEO_SLICEMODE. Ignored incase of IVIDEO_ENTIREFRAME.

Table 4-7. Default and Supported Values for IVIDENC2_DynamicParams

Field	Default Value	Supported Value
size	sizeof(IMPEG4ENC_DynamicParams)	• sizeof(IVIDENC2_DynamicParams) • sizeof(IMPEG4ENC_DynamicParams)
inputHeight	2048	[64, 2048]
inputWidth	2048	[64, 2048]
refFrameRate	30000	Ignore
targetFrameRate	30000	Valid Values as per Level Limit
targetBitRate	8000000	Valid Values as per Level Limit
intraFrameInterval	0	Any value >= 0.
generateHeader	XDM_ENCODE_AU	• XDM_ENCODE_AU • XDM_GENERATE_HEADER
captureWidth	2048	>= inputWidth
forceFrame	IVIDEO_NA_FRAME	• IVIDEO_NA_FRAME • IVIDEO_I_FRAME
interFrameInterval	0	0
mvAccuracy	IVIDENC2_MOTIONVECTOR_HALFPEL	IVIDENC2_MOTIONVECTOR_HALFPEL
sampleAspectRatioHeight	1	Any value, only lower 8 bits are considered by encoder
sampleAspectRatioWidth	1	Any value, only lower 8 bits are considered by encoder
ignoreOutbufSizeFlag	XDAS_TRUE	[0,1]

Field	Default Value	Supported Value
*putDataFxn	NULL	Valid function pointer, NULL
putDataHandle	0	Ignore
*getDataFxn	NULL	Valid function pointer, NULL
getDataHandle	0	Ignore
*getBufferFxn	NULL	Valid function pointer, NULL
getBufferHandle	0	Ignore

Table 4-8. Default and Supported Values for IMPEG4ENC_RateControlParams

Field	Default Value	Supported Value
rateControlParamsPreset	IMPEG4_RATECONTROLPARAMS_DEFAULT	• IMPEG4_RATECONTROLPARAMS_DEFAULT • IMPEG4_RATECONTROLPARAMS_USERDEFINED • IMPEG4_RATECONTROLPARAMS_EXISTING
rcAlgo	IMPEG4_RATECONTROLALGO_VBR	• IMPEG4_RATECONTROLALGO_NONE • IMPEG4_RATECONTROLALGO_VBR • IMPEG4_RATECONTROLALGO_CBR
qpI	5	[1,31]
qpP	5	[1,31]
seInitialQP	5	[0,31]. 0 -> Codec will internally calculate the initial QP required for encoding.
qpMax	31	[1,31]
qpMin	1	[1,31]
enablePerceptualQuantMode	0	[0, 1]
allowFrameSkip	0	[0, 1, 2]
initialBufferLevel	0 -> codec will internally calculate the value based on the rate control algo.	For CBR, minimum value is equal to 0.1 * targetBitRate For VBR, minimum value is equal to 0.4 * targerBitRate. Max value is vbvBufferSize. If user does not know what value to set , then set this value of 0, codec will internally calculate the

Field	Default Value	Supported Value
		value based on the rate control algo.
vbvBufferSize	0 -> codec will internally calculate default value based on rate control algo.	For CBR, minimum value is equal to 0.1 * targetBitRate For VBR, minimum value is equal to 0.6 * targerBitRate. Max value is level dependent. If user does not know what value to set , then set this value of 0, codec will internally calculate the value based on the rate control algo
qpMinIntra	0	[0, 1 to 31] 0 -> disables this feature

Table 4-9. Default and Supported Values for IMPEG4ENC_InterCodingParams

Field	Default Value	Supported Value
interCodingPreset	IMPEG4_INTERCODING_DEFAULT	• IMPEG4_INTERCODING_DEFAULT • IMPEG4_INTERCODING_USERDEFINED
searchRangeHorP	144	[16,144]
searchRangeVerP	32	[16,32]
globalOffsetME	1	[0,1]
earlySkipThreshold	200	[0, 200]
enableThresholdingMethod	1	[0,1]
minBlockSizeP	IMPEG4_BLOCKSIZE_8x8	• IMPEG4_BLOCKSIZE_16x16 • IMPEG4_BLOCKSIZE_8x8
enableRoundingControl	0	[0,1]

Table 4-10. Default and Supported Values for IMPEG4ENC_IntraCodingParams

Field	Default Value	Supported Value
intraCodingPreset	IMPEG4_INTRACODING_DEFAULT	• IMPEG4_INTRACODING_DEFAULT • IMPEG4_INTRACODING_USERDEFINED

Field	Default Value	Supported Value
intraRefreshMethod	IMPEG4_INTRAREFRESH_NONE	• IMPEG4_INTRAREFRESH_NONE • IMPEG4_INTRAREFRESH_CYCLIC_MBS • IMPEG4_INTRAREFRESH_CYCLIC_ROWS • IMPEG4_INTRAREFRESH_MANDATORY
intraRefreshRate	0	>=0, effective only intraRefreshMethod != IMPEG4_INTRAREFRESH_NONE
acpredEnable	1	[0, 1]
insertGOVHdrBeforeIframe	0	[0, 1]
enableDriftControl	1	[0, 1]

Table 4-11. Default and Supported Values for IMPEG4ENC_SliceCodingParams

Field	Default Value	Supported Value
sliceCodingPreset	IMPEG4_SLICECODING_DEFAULT	• IMPEG4_SLICECODING_DEFAULT • IMPEG4_SLICECODING_USERDEFINED • IMPEG4_SLICECODING_EXISTING
sliceMode	IMPEG4_SLICEMODE_NONE	• IMPEG4_SLICEMODE_NONE • IMPEG4_SLICEMODE_MBUNIT • IMPEG4_SLICEMODE_BITS
sliceUnitSize	0	Any value >= 0. If sliceMode is IMPEG4_SLICEMODE_MBUNIT and sliceUnitSize is greater than number of MBs in frame, then it is assumed to be single slice i.e entire frame as single slice. If sliceMode is IMPEG4_SLICEMODE_BITS, then sliceUnitSize indicates number of bits required to form one slice.
gobInterval	0	[0, max MB rows in a frame]
useHec	0	[0, 1, 2]

Table 4-12. Default and Supported Values for IMPEG4ENC_Params

Field	Default Value	Supported Value

Field	Default Value	Supported Value
videnc2Params	See Table 4-6	
rateControlParams	See Table 4-8	
interCodingParams	See Table 4-9	
intraCodingParams	See Table 4-10	
sliceCodingParams	See Table 4-11	
useDataPartitioning	0	[0, 1]
useRvlc	0	[0, 1]
useShortVideoHeader	0	[0, 1]
vopTimeIncrementResolution	30	[1, 65535]
nonMultiple16RefPadMethod	IMPEG4_PAD_METHOD_MPEG4	• IMPEG4_PAD_METHOD_DIVX • IMPEG4_PAD_METHOD_MPEG4
pixelRange	IMPEG4ENC_PR_0_255	• IMPEG4ENC_PR_16_235 • IMPEG4ENC_PR_0_255
enableSceneChangeAlgo	IMPEG4ENC_SCDA_ENABLE	• IMPEG4ENC_SCDA_ENABLE • IMPEG4ENC_SCDA_DISABLE
useVOS	1	[0,1]
enableMONA	0	[0,1]
enableAnalyticinfo	IVIDEO_METADATAPLANE_NONE	• IVIDEO_METADATAPLANE_NONE • IVIDEO_METADATAPLANE_MBINFO
debugTraceLevel	0	• 0 • 1
lastNFramesToLog	0	>= 0

Table 4-13. Default and Supported Values for IMPEG4ENC_DynamicParams

Field	Default Value	Supported Value

Field	Default Value	Supported Value
videnc2DynamicParams	See Table 4-7	
rateControlParams	See Table 4-8	
interCodingParams	See Table 4-9	
sliceCodingParams	See Table 4-11	
aspectRatioIdc	IMPEG4ENC_ASPECTRATIO_SQUARE	• IMPEG4ENC_ASPECTRATIO_SQUARE • IMPEG4ENC_ASPECTRATIO_12_11 • IMPEG4ENC_ASPECTRATIO_10_11 • IMPEG4ENC_ASPECTRATIO_16_11 • IMPEG4ENC_ASPECTRATIO_40_33 • IMPEG4ENC_ASPECTRATIO_EXTENDED

4.4 Interface Functions

This section describes the Application Programming Interfaces (APIs) used in the MPEG4/H.263 Encoder. The 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.

`algNumAlloc()`, `algAlloc()`, `algInit()`, `algActivate()`, `algDeactivate()`, and `algFree()` are standard XDAIS APIs. This document includes only a brief description for the standard XDAIS APIs. For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

4.4.1 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* (literature number SPRU360).

‖ 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. If an algorithm 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* (literature number SPRU360).

‖ See Also

```
algNumAlloc(), algFree()
```

4.4.2 Initialization API

Initialization API is used to initialize an instance of the algorithm. 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; /* algorithm instance handle*/

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.

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

‖ See Also

`algAlloc(), algMoved()`

4.4.3 Control API

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

‖ Name

`control()` – change run time parameters and query the status

‖ Synopsis

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

‖ Arguments

```
IVIDENC2_Handle handle; /* algorithm instance handle */

IVIDENC2_Cmd id; /* algorithm specific control commands*/

IVIDENC2_DynamicParams *params /* algorithm run time
parameters */

IVIDENC2_Status *status /* algorithm instance status
parameters */
```

‖ Return Value

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

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

‖ Description

This function changes the run time parameters of an algorithm instance and queries the algorithm's `status`. 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 an algorithm instance.

The second argument is an algorithm specific control command. See `XDM_CmdId` enumeration 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 and fourth arguments must be pointers to the extended `DynamicParams` and `Status` data structures respectively. Also, ensure that the `size` field is set to the size of the extended data structure. Depending on the value set for the size field, the algorithm uses either basic or extended parameters.

‖ Preconditions

The following conditions must be true prior to calling this function; otherwise, its operation is undefined.

- `control()` can only be called after a successful return from `algInit()` and `algActivate()`.
- If algorithm uses DMA resources, `control()` can only be called after a successful return from `DMAN3_init()`.
- `handle` must be a valid handle for the algorithm's instance object.
- params must not be NULL and must point to a valid `IVIDENC2_DynamicParams` structure.
- status must not be NULL and must point to a valid `IVIDENC2_Status` structure.
- If a buffer is provided in the `status->data` field, it must be physically contiguous and owned by the calling application.

‖ Postconditions

The following conditions are true immediately after returning from this function.

- If the control operation is successful, the return value from this operation is equal to `IALG_EOK`; otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.
- If the control command is not recognized, the return value from this operation is not equal to `IALG_EOK`.
- The algorithm should not modify the contents of params. That is, the data pointed to by this parameter must be treated as read-only.
- If a buffer was provided in the `status->data` field, it is owned by the calling application.

‖ Example

See test application file, TestAppEncoder.c available in the \Client\Test\Src sub-directory.

‖ See Also

`algInit()`, `algActivate()`, `process()`

4.4.4 Data Processing API

Data processing API is used for processing the input data.

‖ 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 encoding/decoding call

‖ Synopsis

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

‖ Arguments

```
IVIDENC2_Handle handle; /* algorithm instance handle */

IVIDEO2_BufDesc *inBufs; /* algorithm input buffer
descriptor */

XDM2_BufDesc *outBufs; /* algorithm output buffer
descriptor */

IVIDENC2_InArgs *inargs /* algorithm runtime input
arguments */

IVIDENC2_OutArgs *outargs /* algorithm runtime output
arguments */
```

‖ Return Value

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

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

‖ Description

This function does the basic encoding/decoding. The first argument to `process()` is a handle to an algorithm instance.

The second and third arguments are pointers to the input and output buffer descriptor data structures respectively (see `XDM_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 an algorithm instance object.

The last argument is a pointer to the `IVIDENC2_OutArgs` data structure that defines the run time output arguments for an algorithm instance object.

> **Note**:
>
> If you are using extended data structures, the fourth and fifth arguments must be pointers to the extended `InArgs` and `OutArgs` data structures respectively. Also, ensure that the `size` field is set to the size of the extended data structure. Depending on the value set for the `size` field, the algorithm uses either basic or extended parameters.

‖ Preconditions

The following conditions must be true prior to calling this function; otherwise, its operation is undefined.

- `process()` can only be called after a successful return from `algInit()` and `algActivate()`.

- If algorithm uses DMA resources, `process()` can only be called after a successful return from `DMAN3_init()`.
- `handle` must be a valid handle for the algorithm's instance object.
- Buffer descriptor for input and output buffers must be valid.
- Input buffers must have valid input data.
- `inBufs->numBufs` indicates the total number of input
- Buffers supplied for input frame, and conditionally, the encoders MB data buffer.
- `inArgs` must not be NULL and must point to a valid `IVIDENC2_InArgs` structure.
- `outArgs` must not be NULL and must point to a valid `IVIDENC2_OutArgs` structure.
- `inBufs` must not be NULL and must point to a valid `IVIDEO1_BufDescIn` structure.
- `inBufs->bufDesc[0].bufs` must not be NULL, and must point to a valid buffer of data that is at least `inBufs->bufDesc[0].bufSize` bytes in length.
- `outBufs` must not be NULL and must point to a valid `XDM2_BufDesc` structure.
- `outBufs->buf[0]` must not be NULL and must point to a valid buffer of data that is at least `outBufs->bufSizes[0]` bytes in length.
- The buffers in `inBuf` and `outBuf` are physically contiguous and owned by the calling application.

‖ Postconditions

The following conditions are true immediately after returning from this function.

- If the process operation is successful, the return value from this operation is equal to `IALG_EOK`; otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.
- After successful return from `process()` function, `algDeactivate()` can be called.
- The algorithm must not modify the contents of `inArgs`.
- The algorithm must not modify the contents of `inBufs`, with the exception of `inBufs.bufDesc[].accessMask`. That is, the data and buffers pointed to by these parameters must be treated as read-only.
- The algorithm must appropriately set/clear the `IVIDEO2_BufDescIn::bufDesc[].accessMask` field in `inBufs` to indicate the mode in which each of the buffers in `inBufs` were read. For example, if the algorithm only read from `inBufs.bufDesc[0].buf` using the algorithm processor, it could utilize `#XDM_SETACCESSMODE_READ` to update the appropriate

accessMask fields. The application may utilize these returned values to manage cache.

- The buffers in inBufs are owned by the calling application.

Example

See test application file, TestAppEncoder.c available in the \Client\Test\Src sub-directory.

See Also

algInit(), algDeactivate(), control()

Note:

- A video encoder or decoder cannot be preempted by any other video encoder or decoder instance. That is, you cannot perform task switching while encode/decode of a particular frame is in progress. Pre-emption can happen only at frame boundaries and after algDeactivate() is called.
- The input data is an uncompressed video frame in one of the format defined by inputChromaFormat of IVIDENC2_Params structure. The encoder outputs MPEG4 compressed bit-stream in the little-endian format.
- outBufs->bufs[0] may contain the encoded data buffer. See IVIDENC2_OutArgs.encodedBufs for more details.
- outBufs->bufs[1], outBufs->bufs[2], and outBufs->bufs[3] are used when providing reconstruction buffers.

‖ 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* (literature number SPRU360).

‖ See Also

`algActivate()`

4.4.5 Termination API

Termination API is used to terminate the algorithm instance 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 MPEG4/H.263 Encoder.

5.1 Release Package

Question	Answer
Can this codec release be used on any Media Controller and HDVICP2 based platform?	Yes, you can use it on any Media Controller and HDVICP2 based platform. But the Test application shipped along with this release is meant for a particular platform. Before using it to different platform, you need to ensure that the addresses provided in linker command file are taken care. In addition, the HDVICP2 related addresses through HDVICP IRES interface should be provided correctly.

5.2 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 setting as File -03 and no debug information which throws an error that code memory section is not sufficient.
Application returns an error saying “Cannot open input file “….YUV” while running the host test app	Make sure that input YUV path is given correctly. If the application is accessing YUVs from network, ensure that the network connectivity is stable.
Codec misbehaves or hangs with some of the features enabled like DPand H.241 on simulator.	This is the known shortcoming in the simulator. These features have been tested and verified on hardware.

5.3 Issues with Tools/FC Version

Question	Answer
What tools are required to run the standalone codec?	To run the codec on standalone 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 Platform specific CSP is needed (See section 2.2 for more details.)

Question	Answer
What tools are required to run the standalone codec?	To run the codec on standalone 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 Platform specific CSP is needed (See section 2.2 for more details.)
Which simulator version should I use for this release of MPEG4 encoder on HDVICP2?	Code Composer Studio (CCSv4) version 4.2.0.09000 has to be installed. DM816x simulator CSP version 0.7.1 (or newer) has to be installed after installing Code Composer Studio, This release can be obtained by software updates on CCSV4. Please make sure that 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
Is this encoder integrated with codec engine, if yes with which version?	Yes, this encoder is integrated with Codec Engine version 3.20.00.16
What CG tools version should I use for code compilation?	You may use CG tools version 4.5.1 to compile the code.
What if the application is using different CG tools version?	The memory layout of the interface data structures does not change with different version of compilers(if bit-fields are not used). In addition, it does not change the mechanism of generating signature for functions. This version can be used even if the application is with different CG tools because no bit-fields are used in interface.

5.4 Algorithm Related

Question	Answer
What XDM interface does codec support?	Codec supports XDM IVIDENC2 interface
Does MPEG4 Encoder support non-multiple of 16 frame height and width?	Yes. But image width should be atleast multiple of 2.
What are the profiles supported in this version encoder?	Supports MPEG4 Simple Profile and H.263 baseline profile.
What is the maximum level supported by this encoder?	Supports MPEG4 Simple Profile levels 0, 0b, 1, 2, 3, 4A, 5 and 6. Supports H.263 baseline profile levels 10, 20, 30, 45, 50, 60 and 70.
The encoder gives error during creation, what could be the reason?	The create call failure is due to non-availability of the memory requested by the codec.
Does this version of MPEG4 Encoder support ASP?	No, Only limited tools of simple profile supported, which is explained in section 4.3
Can DDR footprint of encoder be reduced?	Yes. DDR foot print is dependent on maxWidth and maxHeight parameters and also dependent on feature of Data partitioning.

Question	Answer
What are the input frame formats supported?	This version supports only YUV420 semi-planar input format.
Does this version of MPEG4 Encoder expose motion vectors for a frame to the application?	Yes. Details can be found in Appendix D.
Can I encode with bit rate other than specified for a level in standard?	Yes. Video encoder will return a non-fatal level incompliance error, but still it continues encoding. It is not guaranteed to achieve real time performance for bit rates higher than specified.
Can I control video quality levels?	Yes, but no presets are supported. This is achieved by controlling parameters like globalOffsetME, perceptualQaunt, searchRange, 1MV/4MV, VBV buffer size etc
Does this version of MPEG4 Encoder support interlaced coding?	No, this version of MPEG4 Encoder supports only progressive type coding.
Does this version of MPEG4 Encoder allow me to encode multiple packets?	Yes. In MPEG4 both fixed MBs and fixed bytes per packet is supported where as in short video header only fixed bytes is supported.
Can I disable frac-pel refinement of Motion vectors?	No
Is there a limit on number of slices supported per frame by encoder?	Functionality wise, there is no limit. Encoder can support one slice per MB also. Note, however, that number of slices per frame supported in Real Time depends on frequency of HDVICP2. Ex: For average 40 slices per frame of average content can be supported at 266 Mhz, 1080p resolution, 30 fps.
Can the encoder give multiple Motion vector for a macro block?	Yes
What is the maximum resolution supported by this version of MPEG4 encoder?	This version of MPEG4 encoder supports resolution up to 2048x2048.
Does Algorithm support DataSync mechanism for low-delay applications?	Yes. It has the mechanism for both input and output buffers.
Does Algorithm support H.241 based packetization (slice cap/ maxBitsPerSlice) feature ?	Yes.
Does algorithm support DP with H.241?	Yes. But packet size and bit rate adherence is not guaranteed due to constraint in the hardware.
For a given configuration why performance is poorer incase of H.241 enabled compared to without H.241?	Incase of H.241, for every slice boundaries encoder needs to flush and restart the pipeline t0 meet the strict restriction on the bytes generated for slices. The performance becomes poorer as the number of slices generated per frame is higher (in other words if bytes/ slice is very low).

Question	Answer
Encoder asks few buffers in TILED memory, can I override the encoder's request and provide buffers in different space?	Yes, you can over ride the encoder's request but with below constraints • TILED PAGE can be overridden by RAW • TILED8, TILED16 can be overridden by TILED PAGE, RAW • TILED16 can be overridden by TILED8, RAW, TILED PAGE
Encoder requires large amount of memory to compress bit-streams. The encoder does not require the same amount memory after compression. Can this memory usage be reduced?	Yes, you need to set `igonoreOutBufSizeFlag = XDAS_TRUE && getBufferFxn = Valid Function Pointer` If the application is not capable of providing memory at run time with codec's request by `getBufferFxn` then it can point to a dummy function which returns -1.
Can I change bit-rate, frame rate, resolution at run time	Yes
Can the encoder support higher resolutions at lower FPS?	Yes, But maximum resolution supported is 2048 x2048. Not beyond that.
Is the performance quoted in datasheet the worst-case performance?	No, they are average numbers.
Does the encoder support multi-channel operation?	Yes.
The XDM control call fails, what could be the reason?	The following are few of reasons for the error: • If create time parameter is not set properly then encoder returns back during subsequent process/control call with detailed error code • Encoder is called with un-supported dynamic parameter.
The process call returns error, what are the possible reasons?	The following are few of reasons for the error: • The input or output pointers are null • The input or output buffer sizes are not sufficient or incorrect • Creation/control time failure
Can the motion vectors of a macro block exceeds the search range?	No in case of globalOffsetME is disabled. But with globalOffsetME enabled the search range of a macroblock is with respect to global motion vector and hence MV can exceed the search range.
What is granularity of the process call?	The encoder supports only frame level encoding API. However, it supports data sync APIs for sub frame level data exchange between Application and Encoder, both at input and output side. Refer Appendix for more information.
Can the encoder be run on any OS?	Yes. Encoder implementation is independent of Operating System. Only necessity is that the component interacting with encoder has to be VIDENC2interface compliant.

Question	Answer
Encoder asks few buffers in TILED memory, can I override the encoder's request and provided buffers in different space?	Yes, you can over ride the encoder's request but with below constraints 1. TILED PAGE can be overridden by RAW 2. TILED8, TILED16 can be overridden by TILED PAGE, RAW 3. TILED16 can be overridden by TILED8, RAW, TILED PAGE However note that in cases, 2 and 3, there will be certain performance impacts.
Can Application allocate few Luma buffers in TILED8 and few in other areas (like RAW region)?	No. All Luma Buffers for the given instance of the encoder need to be in same type of area.
Can Application allocate few Chroma buffers in TILED16 and few in other areas (like RAW region)?	No. All Chroma Buffers for the given instance of the encoder need to be in same type of area.

This page is intentionally left blank

Appendix A

H.241 Packetization in Case of H.263

This section describes the mechanism that needs to be taken care by the application to handle the H.263 bit-stream with H.241 packetization.

A.1 Description of the Requirement

In H.263 standard, the packetization concept is restricted to multiple rows of MBs, where in GOB header is inserted by the encoder at beginning of the row. However, incase of real-time low delay network communication, splitting the bit-stream at the MB level whenever the max size of the packet is exceeds is preferred mechanism. This is similar to the Slice Structured mode as described in the ANNEX K of H.263 standard, but with slight modification in the stuffing part. H.241 packetization with H.263 needs the output data synch functionality to be set to SLICE MODE. i.e., outputDataMode = IVIDEO_SLICE_MODE.

Following section describes the mechanism that is needed to handle the packetized H.263 stream to extract the actual bit-stream content from the additional stuffing bits in the bit-stream.

A.2 Bit-stream Stitching Mechanism

The codec library generates the H.263 bit-stream with stuffing bits inserted at the position wherever packet ends at the non byte boundary. This stream cannot be decoded by the decoder due to stuffing bits. Therefore, the application/decoder must stitch the bit-stream between two packets and should be able to extract the actual content of the bit-stream. Following diagram explains the bit-stream structure provided by the codec to the application/decoder.

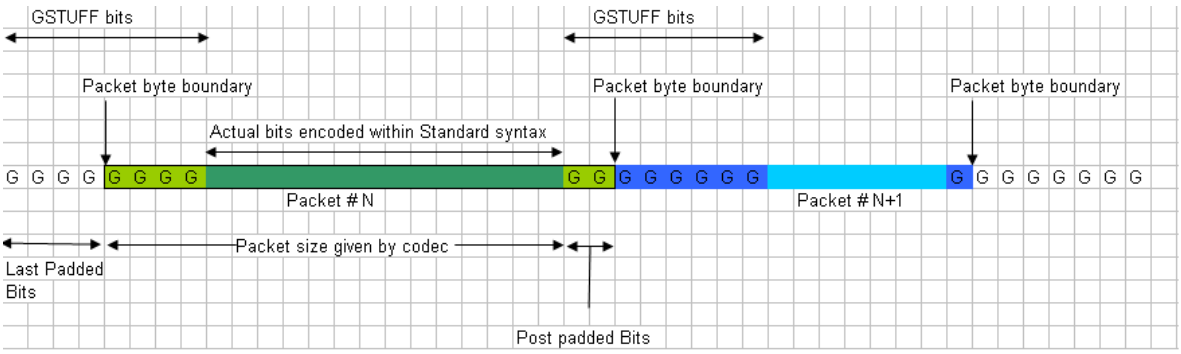


Figure A-1. H.263 stream with GSTUFF bits embedded

Codec embeds the GSTUFF bits in between two packets to make the packet boundary byte aligned. If the packet ends at the byte boundary, then codec does not put any GSTUFF bits. E.g. If the packet N terminated at the non byte boundary say at 2046 bits, then remaining number of bits which required to make byte aligned will be filled by GSTUFF bits i.e. in this case 2 zeros (part of GSTUFF bits) will be inserted in the current packet N. Remaining 6 zeros (other part of GSTUFF bits) will be inserted in the starting of next packet N+1 (refer Figure A-1).

The protocol to communicate the bit-stream details is through DataSync API where in codec gives out number of packets and packet sizes. Note that packet size given out by codec through datasync descriptors will be in-terms of bits not bytes. This will help in recognizing the valid standard bitstream out of GSTUFFed bitstream. If the packet size is multiple of 8, that indicates packet is ended at the byte boundary and no stuffing bits are put by the codec and size of the packet size will be in the data sync descriptors.

Application/decoder is required to remove these GSTUFF bits and extract only standard syntax bits and stitch the same with next packet to generate the final standard syntax bits.

Appendix B

Debug Trace Usage

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

B.1 Introduction

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

The primary use of Debug Trace Usage are:

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

  – 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 behaviour

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.

B.2 Enabling and Using Debug Information

To enable debug information, following two parameters are added in the create time parameter list .

– `debugTraceLevel`

  – `lastNFrameToLog`

Hence, the MPEG4 Encoder create time parameter is modified as:

```
typedef struct IMPEG4ENC_Params {
  IVIDENC2_Params videnc2Params;
  IMPEG4ENC_RateControlParams rateControlParams;
  IMPEG4ENC_InterCodingParams interCodingParams;
  IMPEG4ENC_IntraCodingParams intraCodingParams;
  IMPEG4ENC_sliceCodingParams sliceCodingParams;
  XDAS_UInt32 useDataPartitioning;
  XDAS_UInt32 useRvlc;
  XDAS_UInt32 useShortVideoHeader;
  XDAS_UInt32 vopTimeIncrementResolution;
  XDAS_UInt32 nonMultiple16RefPadMethod;
  XDAS_UInt32 pixelRange;
  XDAS_UInt32 enableSceneChangeAlgo;
```

```
  XDAS_UInt32 useVOS;
  XDAS_UInt32 enableMONA;
  XDAS_Int32  enableAnalyticinfo;
  XDAS_UInt32 debugTraceLevel;
  XDAS_UInt32 lastNFramesToLog;
} IMPEG4ENC_Params;
```

B.2.1 debugTraceLevel

This parameter configures the codec to dump a debug trace log.

- 0: Disables dumping of debug trace parameters
- > 0: ,Enables dumping of debug trace parameters, value specifies the debug trace level

This parameter can take on additional values in future to support higher and higher levels of debug trace

B.2.2 lastNFramesToLog:

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

- 0: Current frame will be maintained by the codec
- > 0: History of past specified number of frames + Current frame 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
{
  IVIDENC2_Status videnc2Status;
  IMPEG4ENC_RateControlParams rateControlParams;
  IMPEG4ENC_InterCodingParams interCodingParams;
  IMPEG4ENC_IntraCodingParams intraCodingParams;
  IMPEG4ENC_sliceCodingParams sliceCodingParams;
  XDAS_UInt32 useDataPartitioning;
  XDAS_UInt32 useRvlc;
  XDAS_UInt32 useShortVideoHeader;
  XDAS_UInt32 vopTimeIncrementResolution;
  XDAS_UInt32 nonMultiple16RefPadMethod;
  XDAS_UInt32 pixelRange;
  XDAS_UInt32 enableSceneChangeAlgo;
  XDAS_UInt32 useVOS;
  XDAS_UInt32 enableMONA;
  XDAS_Int32  enableAnalyticinfo;
  XDAS_UInt32 debugTraceLevel;
  XDAS_UInt32 lastNFramesToLog;
```

```
  XDAS_UInt32 *extMemoryDebugTraceAddr;
  XDAS_UInt32 extMemoryDebugTraceSize;
} IMPEG4ENC_Status;
```

`debugTraceLevel`: Debug trace level configured for the codec - 0, 1

`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.

B.3 Requirements on the Application Side

The following are the requirements on the application side:

– The application should be capable of configuring `debugTraceLevel` and `lastNFrameToLog` which are part of the Initialization Parameters of the codec

  – The application should be capable of querying the codec for its debug parameter memory regions and size

  – 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.

  – 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

Appendix C

DataSynch API Usage Guide

This section describes the data synch API usage from application point of view with codec.

C.1 Description

Most of the TI Video Codec interfaces prior to IVIDENC2 and IVIDDEC3 allow only 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 needed then it is via codec's extended interface. This document explains the sub-frame level data communication capabilities of video codec using data synch call backs defined with IVIDENC2 interface.

C.2 Video Encoder Input with Sub-frame Level Synchronization

This section explains the IVIDENC2 interface details, which help to achieve the sub-frame level communications.
Table C-1, Table C-2 and Table C-3 explain the creation, control and handshake parameters related to sub frame level data communication for input data of video encoder respectively.
Details column is a generic column and "valid values" column is specific to video encoder input.

Table C-1.Creation time parameter related to sub frame level data communication for input-data of video 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 if `inputDataMode == IVIDEO_ENTIREFRAME` Any positive value if `inputDataMode == IVIDEO_NUMROWS`

Table C-2. Dynamic Parameters Related to sub–frame Level Data Communication for Input Data of Video Encoder

Parameter Name	Details	Valid values
IVIDENC2_DynamicParams::getDataFxn	This function pointer is provided by the app/framework to the video 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. 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 app-registered IVIDENC2_DynamicParams.getDataFxn(). The app/framework can use this handle to differentiate callbacks from different algorithms.	Any Value

Table C-3. Handshake Parameters Related to Sub-frame Level Data Communication for Input Data of Video 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	Don't care since it is assumed to be contiguous yuv buffer and initial address is via inbuf.

Parameter Name	Details	Valid values
	directly to the start of the first block, and is not treated as a pointer to an array. If `scatteredBlocksFlag` is set to `XDAS_TRUE`, this field points to an array of pointers to the data blocks.	
`XDM_DataSyncDesc::numBlocks`	Number of data blocks	Any Value. If <= zero then codec assumes no data provided and does call back to App again. The unit of this is number of row.
`XDM_DataSyncDesc::varBlockSizeFlag`	Flag indicating whether any of the data blocks vary in size.	Don't care, as unit of size is one row
`XDM_DataSyncDesc::blockSizes`	Variable block sizes array.	Don't care Since unit is assumed to be multiple of number of rows which is indicated by `numBlocks`.

If application want to use video encoder to operate with sub frame on input side

- It should create the video 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.

C.3 Video Encoder Output with Sub-frame Level Synchronization

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

Table C-4, Table C-5 and Table C-6 explain the creation, control and handshake parameters related to sub frame level data communication for output data of video encoder respectively.

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

Table C-4. Creation Time Parameter Related to Sub-frame Level Data Communication for Output Data of Video Encoder

Parameter Name	Details	Valid values
`IVIDENC2_Params::outputDataMode`	Defines the mode of providing the output data.	`IVIDEO_ENTIREFRAME` : Entire frame bit-stream 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 2K. `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 2K) at which encoder should inform the application. For example: Here 4 means that encoder should inform after producing every 4*2048 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 C-5. Dynamic parameters related to sub frame level data communication for output data of video encoder

Parameter Name	Details	Valid values
`IVIDENC2_DynamicParams::putDataFxn`	This function pointer is provided by the app/framework to the video 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-NUL	Any non-NULL value if `outputDataMode != IVIDEO_ENTIREFRAME`
`IVIDENC2_DynamicParams::putDataHandle`	It defines the handle to be used while informing data availability to	Any Value

Parameter Name	Details	Valid values
	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 app-registered `IVIDENC2_DynamicParams.putDataFxn()`. The app/framework can use this handle to differentiate callbacks from different algorithms.	

To simplify the codec implementation, the information sharing by codec to application happens at a quantum of 2K byte data. In this document each 2K byte, is referred as page.

C.3.1 For outputDataMode Equal to IVIDEO_SLICEMODE

Incase of `outputDataMode = IVIDEO_SLICEMODE`, following points should be noticed

1. `numOutputDataUnit` is the frequency after which codec will inform to Application. So in `IVIDEO_SLICE_MODE`, lets `numOutputDataUnit` is 8 then after 8 slices, codec has to make `putData` call. Larger the number of `numOutputDataUnit`, 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 `numOutputDataUnit` as 8.
2. Within 2K, encoder can not handle generation of more than 8 slices (it is specific constraint of the TI's encoder implementation to keep SL2 size impact minimal, as each data unit requires SL2 space to store the associated information).
3. Information point to app (i.e. point at which codec makes putData call) is at 2K boundary.
4. Bit-stream is assumed to be contiguous in memory incase of packet generation based on MBs. This is due to the fact that slice size cannot be assumed to be exact during creation time incase of MBs based slice, hence the bit-stream address is obtained during process call and the `XDM_DataSyncDesc::baseAddr` is don't care for `sliceMode = IMPEG4_SLICEMODE_MBUNIT`. It is a constraint of TI's encoder implementation.
   For `sliceMode = IMPEG4_SLICEMODE_BITS`, both contiguous or non-contiguous memory is supported. For details on non-contiguous memory support, refer to the section C.4.
5. 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 video encoder for more details, section C.4)

6. 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 C-6. Handshake parameters related to sub frame level data communication for output data of video 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.	Flag indicating whether the individual data slices may be scattered in memory. Constraint: This will be `XDAS_FALSE for sliceMode = IMPEG4_SLICEMODE_MBUNIT` and will be `XDAS_FALSE or XDAS_TRUE for sliceMode = IMPEG4_SLICEMODE_BITS`.
`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 to an array of pointers to the data blocks. 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	Any Value and it is the number of slices generated till the point of `putData` call. If `outputDataUnit` is 7, in the page cross over which would be the communication point and it generated 8 slices, then `numbBlocks` is 8 and all 8 slices will be informed to App. `1 <= numBlocks <= 15`
`XDM_DataSyncDesc::varBlockSizeFlag`	Flag indicating whether any of the data blocks vary in size.	XDAS_TRUE or XDAS_FALSE(slice sizes are not constant most of the time)

Parameter Name	Details	Valid values
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]). *Note : Incase of H.263 enabled with H.241, the blockSizes given out by codec will be interms of bits not bytes. Please refer Appendix A for more details.*

C.3.2 For outputDataMode equal to IVIDEO_FIXEDLENGTH

Incase of outputDataMode = IVIDEO_FIXEDLENGTH, following points should be noticed

1. numOututDataUnit is the frequency after which codec will inform to App. so in IVIDEO_FIXED_LENGTH, lets outputDataUnit is 10 then after 20KB codec will make putData call. if numOutputDataUnit is 10, and initial bit-stream buffer size given in process call is 0.5 KB, then codec will put a putData call after 18.5 kB of encoding, not after 21.5 kB.
2. 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 video encoder for more details)
3. 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 C-7. Handshake parameters related to sub frame level data communication for output data of video 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 slices may be scattered in memory. XDAS_TRUE or XDAS_FALSE
XDM_DataSyncDesc::baseAddr	Base address of single data block or pointer to an array of data block addresses of size numBlocks. If scatteredBlocksFlag is set to XDAS_FALSE, this field points directly to the start of the first block, and is not treated as a pointer to an array. If scatteredBlocksFlag is set to XDAS_TRUE, this field points to an array of pointers to the data blocks.	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 to an array of pointers to the data blocks. 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])

If application want to use video encoder to operate with sub frame on input side,

1. It should create the video encoder with `IVIDENC2_Params::outputDataMode = IVIDEO_SLICEMODE` or `IVIDEO_FIXEDLENGTH`.
2. It should also make a control call with `IVIDENC2_DynamicParams::putDataFxn = non-NULL;` to use sub frame level data communication, control call is mandatory.
3. 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.

C.4 Video Encoder with partial buffer on output side

With IVIDENC2 interface video 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`.

Application also need to set `IVIDENC2_DynamicParams::ignoreOutbufSizeFlag` as true to prevent encoder reporting error.

Table C-8 and Table C-9 explain the control and handshake parameters related to sub frame level data communication to handle partial output buffer by video encoder respectively.

Details column is a generic column and “valid values” column is specific to video encoder.

Table C-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 video 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	Any Value

Parameter Name	Details	Valid values
	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.	

Table C-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, if NULL then again call back. If baseAddress[i] is NULL then again call back (where i=0 to numBlock -1 when scatteredBlocksFlag is non-zero)
XDM_DataSyncDesc::numBlocks	Number of data blocks	Any Value. If <= zero then codec assumes no data provided and does call back to App again. <=8 if scatteredBlocksFlag != 0
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.	non-NULL. If it is NULL then again call back blockSizes[i] is terated as blockSizes[i] && 0xFFFFF800. This is to make sure that the block size is always multiple of 2K totalBlockSize = SUM(blockSizes[0] to blockSizes[numBlocks-1]) if

Parameter Name	Details	Valid values
		varBlockSizesFlag is non zero. totalBlockSize = numBlocks * blockSizes[0] if varBlockSizesFlag is zero if totalBlocksSize is 0 the call back again

Following points should be noticed to use video encoder with partial buffer on output side

1. 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
2. For non slice mode and sliceMode = IMPEG4_SLICEMODE_MBUNIT,
    a. During process call the initial stream address and size are provided by application. No constraint on this information and encoder consumes this buffer space.
    b. During data synch (via getBuf) codec can accept a multiple non contiguous buffers from application each of them has to be multiple of 2K. This is true for non slice mode. But incase of sliceMode = IMPEG4_SLICEMODE_MBUNIT, memory cannot be non-continuos as slice sizes are not deterministic and can fall same slice in different memory regions incase non-continous memory is given .
    c. Codec assumes the lower 11 bits to be 0 for the size to make sure that size is multiple of 2K bytes.
3. For sliceMode = IMPEG4_SLICEMODE_BITS (H.241 mode),
    a. For non-continuous stream buffer , during the process call, the initial buffer size should be atleast equal to slice size or multiple of slice size i.e. blockSize = N x sliceUnitSize, where N can take any value greater than zero.

       Assumption here is that, in one getBuf call codec expects blockSize such that for one complete slice stream buffer is continuous. E.g. if slice size is 600 bytes, then in one getBuf call, blockSize should be atleast 600 bytes and should be continuous for entire 600 bytes.

       Application can give different DDR address in each getBuf call meaning each slice can sit in different memory locations in DDR stream buffer.
    b. For continuos stream buffer, refer to point 2)
4. 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
5. If scatteredBlocksFlag flag is zero than there is no limit on numBlocks.

6. 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 (`XDM_INSUFFICIENT_DATA`).

This page is intentionally left blank

Appendix D

Motion Vector and SAD Access API

This section describes the method to access MV and SAD (Analytic Information) data dumped by the encoder.

D.1 Description

The Motion Vector and SAD Access API is a part of the XDM `process()` call, used by the application to encode a frame. A parameter `enabledAnalyticinfo` is provided as a part of create time parameters, which can be set or reset at a frame level during create-time. Setting this flag to 1 indicates that the analytic info is needed. When this parameter is set to 1, the `process()` call returns the motion vector and SAD data in the buffer provided by the application.

For every macro block, the data returned is 8 bytes, a signed horizontal displacement component (signed 16-bit integer) and a vertical displacement component (signed 16-bit integer) and SAD, as shown.

The following sequence should be followed for Analytic Info access:

1. In the create time parameters, set the flag to access analytic data.

```
/* Enable MV access */
createParams ->enableAnalyticinfo =
IVIDEO METADATAPLANE MBINFO;
```

2. Allocate output buffers and define the output buffer descriptors

```
/* Output Buffer Descriptor variables */
XDM BufDesc  outputBufDesc;


/* Get the input and output buffer requirements for the
codec */
control(.., XDM_GETBUFINFO, extn_dynamicParams, ..);
```

If Analytic info access is enabled in step1, this call returns the output buffer info as `numBufs =2,` along with the minimal buffer sizes.

```
/* Initialize the output buffer descriptor */
outputBufDesc.numBufs = 2;
/* Stream Buffer */
outputBufDesc.bufs[0]     = streamDataPtr; //pointer to
mpeg4 bit-stream
```

```
outputBufDesc.bufSizes[0] =
status.bufInfo.minOutBufSize[0];


/* MV Buffer */

outputBufDesc.bufs[1]     =  Output Buffer Base Addr;
//pointer to MV data

outputBufDesc.bufSizes[1] =
status.bufInfo.minOutBufSize[1];
```

3. Call frame encode API

   ```
   /* Process call to encode 1 frame */

   process(..  ,..  , outputBufDesc, .. );
   ```

   After this call, the buffer `outputBufDesc.bufs[1]` will have SAD and Motion vector data.

The data format of output buffer given by user for MV/SAD exposure should be like,

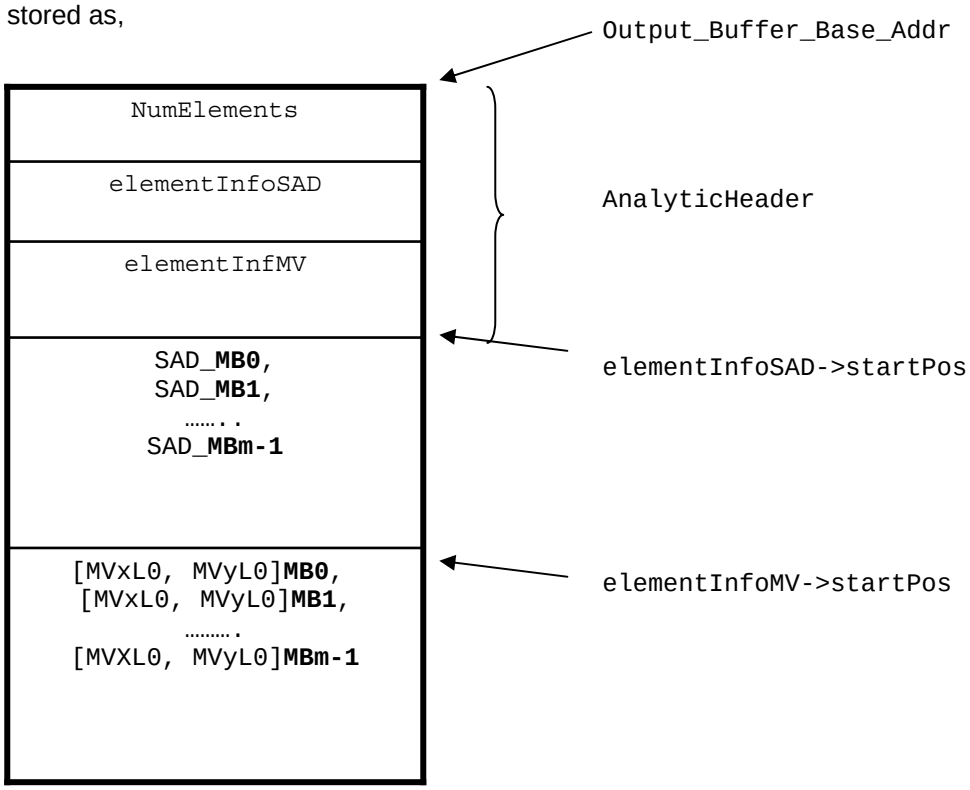
AnalyticHeaderInfo	Data (MV and SAD)

Define a structure:

```
struct AnalyticHeaderInfo

{

       U32 NumElements;

       ElementInfo elementInfoSAD;

       ElementInfo elementInfMV;

} ;


Where as

NumElements -> Total number of elements in the buffer
(As of now   SAD and MV)

ElementInfo is

typedef struct

{

  /*Starting position of data from the buffer base
address*/

  U32 StartPos;

                                     /* No. of bytes to
jump from the current position to get the next data

        of this element group */

  U16 Jump;

    /* Number of data elements in this group */

  U16 Count;
```

```
}Element_Info;
```

For example, consider the data in output buffer dumped by the codec is stored as,



To get the MVL0 data for all macroblocks, the application should have code as,

```
S16 *Src = (U32)Output_Buffer_Base_Addr +
    elementInfoMV->StartPos;
U16 Jump = elementInfoMV->Jump;
S16 *MVL0 = Addr_to_store_MV_inL0

for (i = 0; i < elementInfoMV->Count; i = i++)
{
   * MVL0 ++ = Src[i * Jump]; // To get MVx
   * MVL0 ++ = Src[((i *Jump) + 1)]; //To get MVy
}
```

Note:

- The motion vectors are with halfpel resolution. The halfpel resolution MVs are the fulpel MVs multiplied by 2 and added with the halfpel MVs in the range [+/-1,+/-1].
- SAD = ABS(Ref(i,j) – Src(i,j)) where, Ref is the macro block of the reference region and Src is the macro block of the source image.
- The motion vectors seen in the encoded stream is based on the best

coding decision, which is a combination of motion estimation and mode decision. The MV buffer returns the results of the motion estimation in halfpel resolution (lowest SAD) and this may be different from the motion vectors seen in the bit-stream. More details are given below :

Chapter 6Some macro blocks in a P-frame may be coded as Intra macro blocks based on the post motion estimation decisions. In this case, the motion vectors computed in the motion estimation stage (assuming that this macro block is inter) is returned.

Chapter 7Due to the post motion estimation decisions for some macro blocks, the actual motion vector encoded may be forced to (0,0). In this case, the non-zero motion vector available after the motion estimation is returned.

Chapter 8Some inter macroblocks may be “not coded” due to zero residual. In this case, the half pel motion vectors computed in the motion estimation stage are returned.

Chapter 9For I-frames, motion vectors and SAD in the buffer will be zero.

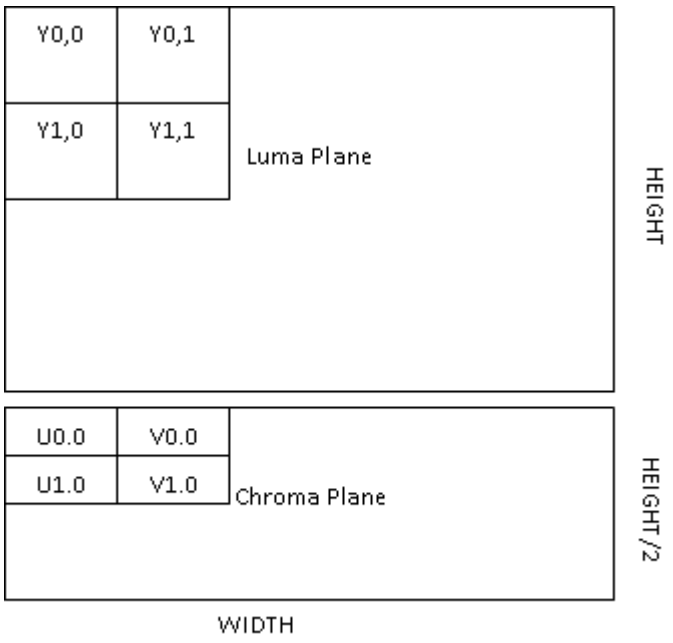
Appendix E

Picture Format

This appendix explains the picture format details for encoder. Encoder expects the input uncompressed picture to be in NV12 format.

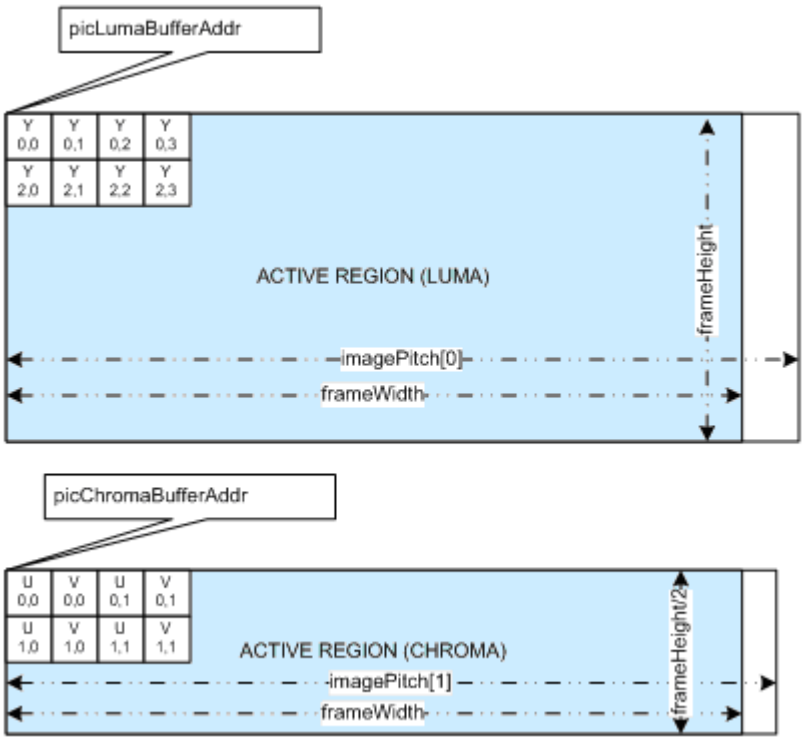
E.1 NV12 Chroma Format

NV12 is YUV 420 planar with 2 separate planes, one for Y, one for U and V interleaved.



E.2 Progressive and Interlaced Format

E.2.1 Progressive Format

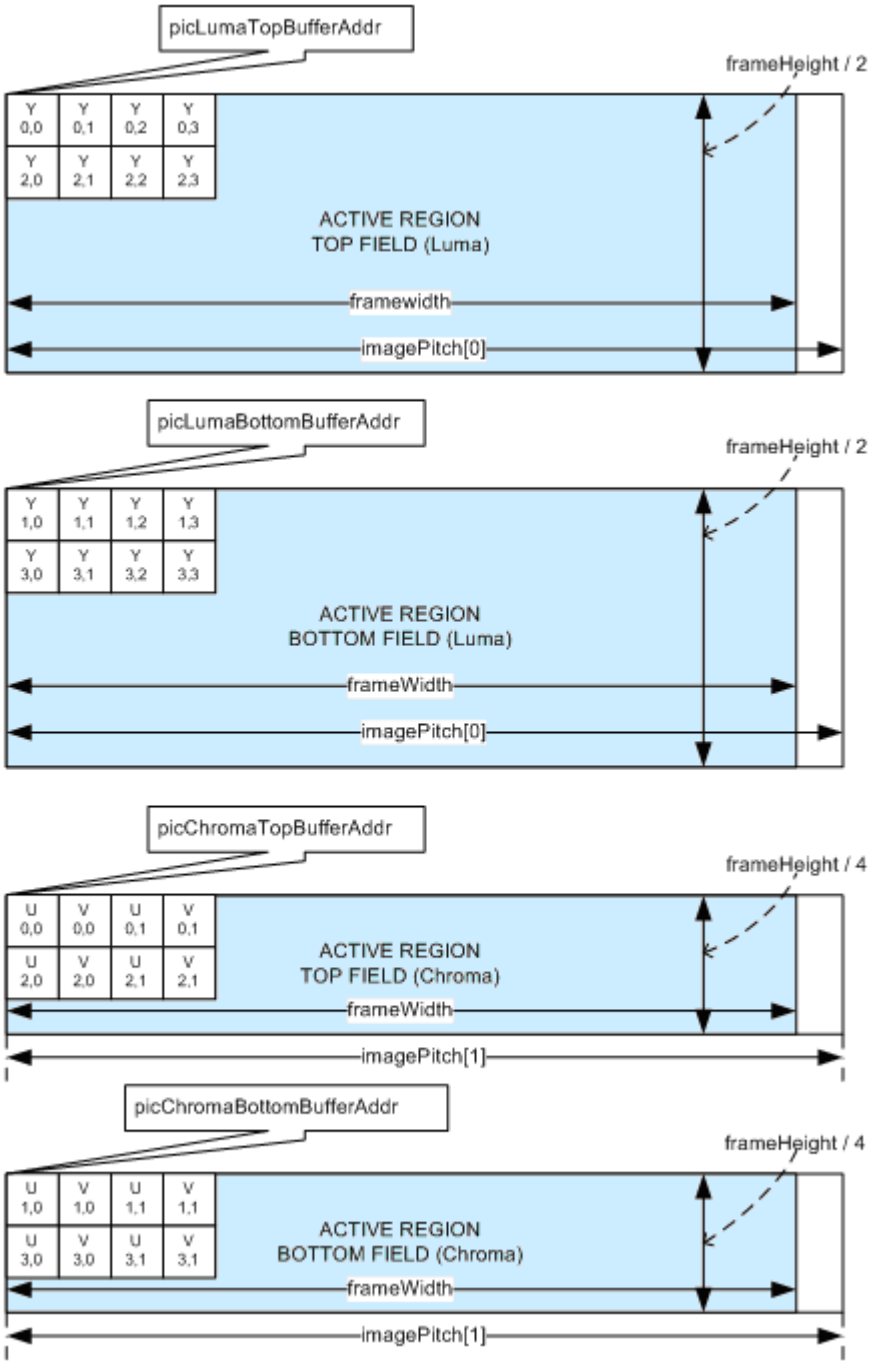


ActiveRegion: Data to be encoded

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.

E.2.2 Interlaced 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

The figure shown is for the case when field data is separate,

E.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
- In interlaced picture for field interleaved case the luma and chroma buffer must be in RAW buffer

Constraints on resolutions are defined as below

- Progressive

  Minimum frameWidth = 96

  Minimum frameHeight = 80

  Maximum frameWidth = 2048

  Maximum frameHeight = 2048

  frameWidth shall be a multiple of 16 bytes

  frameHeight shall be multiple of 2

- Interlaced

  Minimum frameWidth = 96

  Minimum (frameHeight/2) = 80

  Maximum frameWidth = 2048

  Maximum (frameHeight/2) = 2048

  frameWidth shall be a multiple of 16 bytes

  frameHeight shall be multiple of 4.