H.264 Encoder 2.0 on HDVICP2 and Media Controller Based Platform

User's Guide

Literature Number: SPRUHG3
October 2012

This page is intentionally left blank

IMPORTANT NOTICE

Products		**Applications**	
Audio	www.ti.com/audio	Automotive & Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications & Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers & Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energyapps
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video & Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

This page is intentionally left blank

Preface

Read This First

About This Manual

This document describes how to install and work with Texas Instruments' (TI) H.264 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 few frequently asked questions related to using this encoder.
- **Appendix A - Meta Data Support,** explains the meta data support by encoder.
- **Appendix B - Control for Configurable NALU,** explains the configurable NAL unit support by encoder.

- **Appendix C - Control for User Defined Scaling Matrices,** explains the mechanism of supporting user defined scaling matrices.
- **Appendix D - Motion Vector and SAD Access API,** describes the method to access MV and SAD (Analytic Information) data dumped by the encoder.
- **Appendix E – Debug Trace Support,** describes the method to use H.264 encoder debug and trace mechanism.
- **Appendix F – Picture Format,** describes the different format of uncompressed video, which are supported by encoder and the constraints
- **Appendix G – Low Latency / Sub Frame Level Synchronization,** describes the method to achieve ultra low latency on the input and output side of video encoder.
- **Appendix H – Long Term Reference Picture Schemes,** describes the method to get long-term reference picture schemes to get error resilient compressed bit-stream.
- **Appendix I – Hierarchical P Structure Coding Scheme**, describes the method of Hierarchical P structure coding scheme to get bit-stream which has flexibility to have a scalable bitstream in terms of bitrate and framerate without adding any additional dealy.
- **Appendix J – Mapping of Encoding Presets,** describes the method to use extended parameters values that user need to set to meet the exact behaviour of a particular encoding preset
- **Appendix K – Region of Interest Encoding,** describes the method to enable and use the Region of Interest feature.
- **Appendix L – Watermarking SEI Message,** provides information on the support for watermarking in this encoder
- **Appendix M – N Frame Process Call Support,** describes the usage of N frame processing in single process call.
- **Appendix N – Rate Control - High Fidelity Variable Bitrate,** provides an insight to the High Fidelity Variable Bitrate (HF-VBR) Rate Control details of the encoder.
- **Appendix O – Gradual Decoder Refresh (GDR)** - an Error resilience Feature, provides a brief understanding usage details of GDR.

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 11172-2 Information Technology -- Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbits/s -- Part 2: Video (MPEG-1 video standard)
- ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC - Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification

Abbreviations

The following abbreviations are used in this document.

Table 1-1. List of Abbreviations

Abbreviation	**Description**
AIR	Adaptive Intra Fresh
API	Application Programming Interface
AVC	Advanced Video Coding
BP	Base Profile
CAVLC	Context Adaptive Variable Length Coding
CIF	Common Intermediate Format
COFF	Common Object File Format
DMA	Direct Memory Access
DMAN3	DMA Manager
DSP	Digital Signal Processing
EVM	Evaluation Module
GDR	Gradual Decoder Refresh
GOP	Group Of Pictures
HEC	Header Extension Code
HPI	Half Pixel Interpolation
IDR	Instantaneous Decoding Refresh
IRES	Interface for Resources
LTRP	Long Term Reference Picture
NAL	Network Abstraction Layer
PPS	Picture Parameter Set
QCIF	Quarter Common Intermediate Format
QP	Quantization Parameter
QVGA	Quarter Video Graphics Array
RMAN	Resource Manager

Abbreviation	Description
SPS	Sequence Parameter Set
SQCIF	Sub Quarter Common Intermediate Format
SVC	Scalable Video Coding
VGA	Video Graphics Array
XDAIS	eXpressDSP Algorithm Interface Standard
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 (H.264 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

Code Composer Studio, DSP/BIOS, eXpressDSP, TMS320, TMS320C64x, TMS320C6000, TMS320DM644x, and TMS320C64x+ are trademarks of Texas Instruments.

All trademarks are the property of their respective owners.

This page is intentionally left blank

Contents

This page is intentionally left blank

Figures

This page is intentionally left blank

Tables

This page is intentionally left blank

Chapter 1

Introduction

This chapter provides a brief introduction to XDAIS and XDM. It also provides an overview of TI's implementation of the H.264 Encoder on the HDVICP2 and Media Controller Based Platform and its supported features.

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

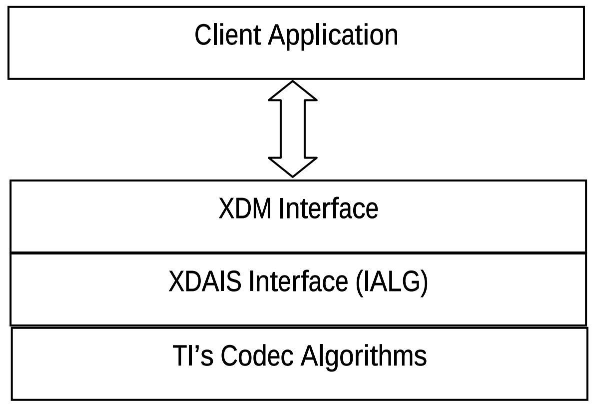
In the multimedia application space, you have the choice of integrating any codec into your multimedia system. For example, if you are building a video encoder system, you can use any of the available video encoders (such as MPEG4, H.263, or H.264) in your system. To enable easy integration with the client application, it is important that all codecs with similar functionality use similar APIs. XDM was primarily defined as an extension to XDAIS to ensure uniformity across different classes of codecs (for example audio, video, image, and speech). The XDM standard defines the following two APIs:

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

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

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

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



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

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

1.1.3 IRES Overview

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

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

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

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

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

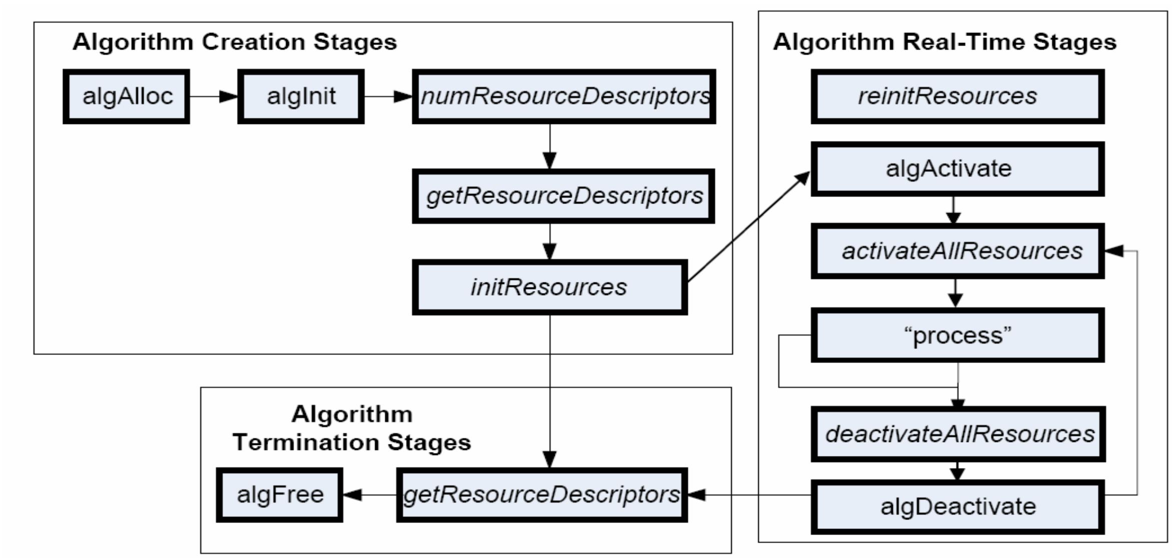


Figure 1-1. IRES Interface Definition and Function Calling Sequence.

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

1.2 Overview of H.264 Encoder

H.264 is the latest video compression standard from the ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group. H.264 provides greater compression ratios at a very low bit-rate. The new advancements and greater compression ratios available at a very low bit- rate has made devices ranging from mobile and consumer electronics to set-top boxes and digital terrestrial broadcasting to use the H.264 standard. Figure 1-2 depicts the working of the H264 Encoder algorithm.

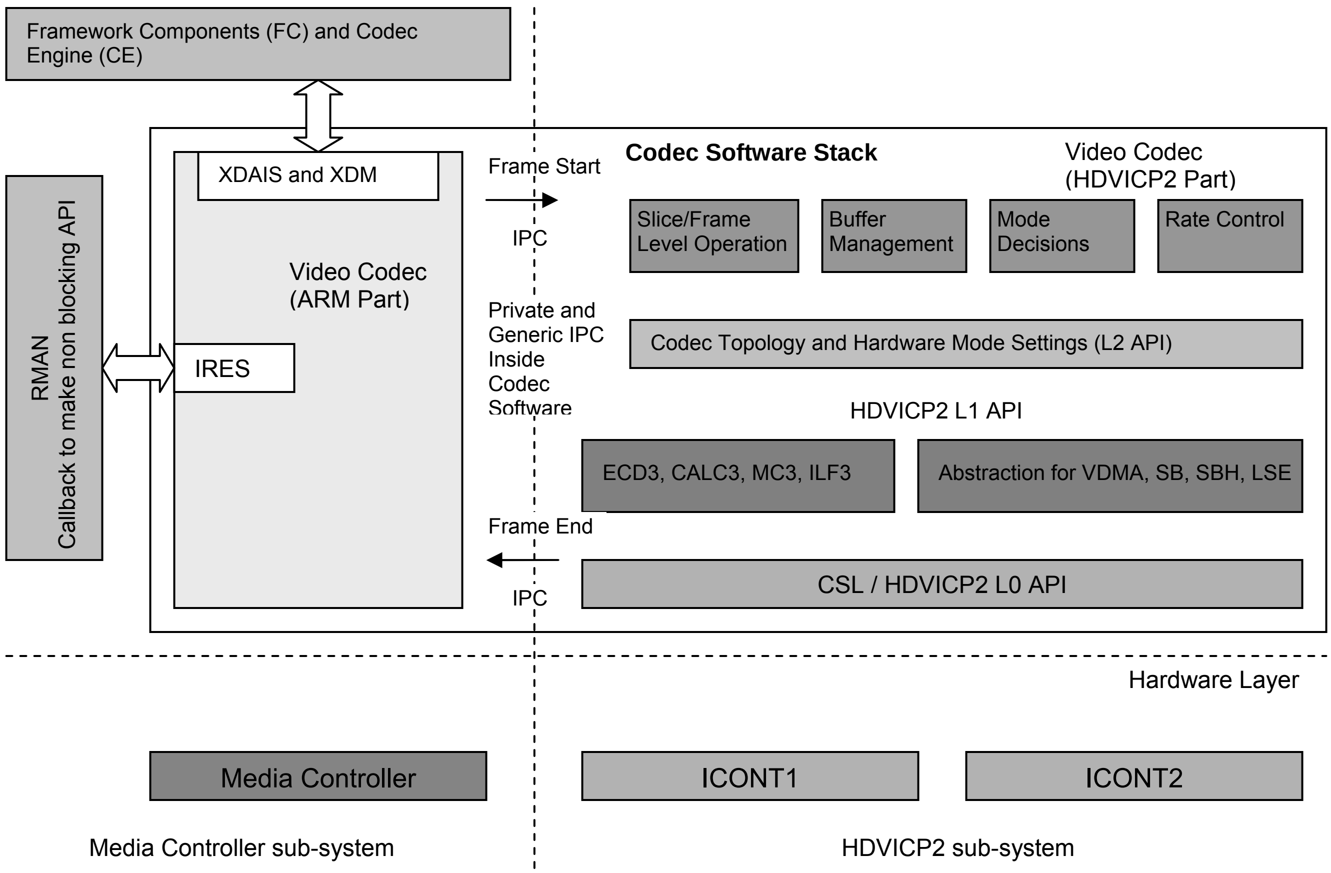


Figure 1-2. Working of H.264 Video Encoder

H.264 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
- 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 needs belongs to this part. This part of the video codec is exposed to system software and core part is hidden.

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 H.264 Encoder on the HDVICP2 and Media Controller Based Platform.

This version of the codec has the following supported features of the standard:

- Supports H.264 baseline, high and main profile up to level 5.1
- Supports arbitrary resolution from 96x80 to 4352 x 4096. Encoder should be created with appropriate level – for example, Level 5.1 for 4096x2048.
- Supports stereoscopic SEI for 3D video coding
- Supports B frame encoding
- Supports progressive and interlaced coding with different controls such as ARF (Adaptive Reference Field), MRF (Most recent Reference Field), and SPF (Same parity Reference Field)
- Supports multiple Scaling Matrix Preset and User Defined Scaling Matrices
- Supports Region of Interest (ROI) encoding along with privacy masking capability. Maximum number of regions supported is 36
- Supports SVC Temporal scalability and Hierarchical-P coding with maximum of 4 temporal layers
- Supports Hierarchical-P field based interlaced coding with different controls such as MRF (Most Recent Reference Field) and SPF (Same parity reference field) with maximum of 4 temporal layers
- Supports Multi frame processing capability in single process call
- Supports watermarking of encoded data for tamper detection.
- Supports different error resilient features like Gradual Decoder refresh, Long term Reference picture Encoding, Cyclic intra refresh mechanism, constrained intra prediction.
- Supports H264 Lite configuration(High Speed preset) for Higher performance
- Supports long term reference frame and allows user to force referencing to long term reference frame at frame level to improve error resilience capability
- Supports insertion of IDR frame at random point with forceFrame control
- Supports user controlled partition size till 8x8 block for inter prediction
- Supports all user controlled POC types: 0, 1 and 2
- Supports low latency features – sub frame level synchronization for input and output. Output data synchronization is based upon slices and fixed length of bit-stream and input data synchronization is based on MB rows.
- Supports change of resolution, frame rate, bit rate and a lot of other parameters dynamically

- Supports TI propriety rate control for storage and low delay devices with finer control of quantization parameter range, initial Quantization Parameter, HRD Buffer Size, max and min Pic Size, Partial Frame Skip, MB level perceptual Rate control and expensive coefficients threshold
- Supports masks to insert user controlled NALU at different access points in the sequence
- Supports Encoding SEI messages containing GMV and RefIdx information to enable closed loop decoder
- Supports forcing a frame or field pair with all macroblocks as skipped
- Supports multiple slices per picture based upon number of macroblocks in each slice or sliceStartOffset
- Supports multiple slices per picture based upon number of bytes per slice for H.241 based MTU packetization
- Supports H.241 defined RCDO profile and staticMbCount exposure
- Supports user controlled in-loop filtering
- Supports exposure of Analytic Info – SAD and motion vector
- Supports image width and height that are multiple of 16, also supports image height being non-multiple of 16
- Supports user controlled quarter-pel interpolation and integer pel for motion estimation
- Supports unrestricted motion vector search that allows motion vectors to be outside the frame boundary
- Supports user controlled all intra modes (4x4, 16x16, and 8x8)
- Supports user controlled constraint set flags
- Supports 8x8 and 4x4 transform size
- Supports user controlled IDR frequency control
- Supports buffering period, timing_info, stereo video info SEI and user defined SEI
- Supports control to have Bottom field first for interlaced coding
- Supports control to have Bottom field Inter or Intra for interlaced coding
- Supports user configurable Group of Pictures (GOP) length and different GOP structures: Non-Uniform (IBBP) and Uniform (BBIBBP)
- Supports control to enable/disable skip MB
- Supports capability to generating only headers

The other explicit features that TI's H.264 Encoder supports are

- eXpressDSP Digital Media (XDM IVIDENC2) interface compliant
- Supports multi-channel functionality
- Supports booting of HDVICP2
- Implements different power optimization schemes
- Supports YUV 420 semi-planar color subsampling format
- Independent of any operating system
- Ability to get plugged in any multimedia frameworks (eg. Codec Engine, OpenMax, GStreamer, etc)

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.

2.1 System Requirements

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

2.1.1 Hardware

This codec has been built and tested on the 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.

  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.

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

  Even you receive 4.5.1 CG tools with CCSV4 installation, please install again by taking from the above link

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

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.H264AVC.E.IVAHD.02.00, under which directory named IVAHD_001 Is created

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

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

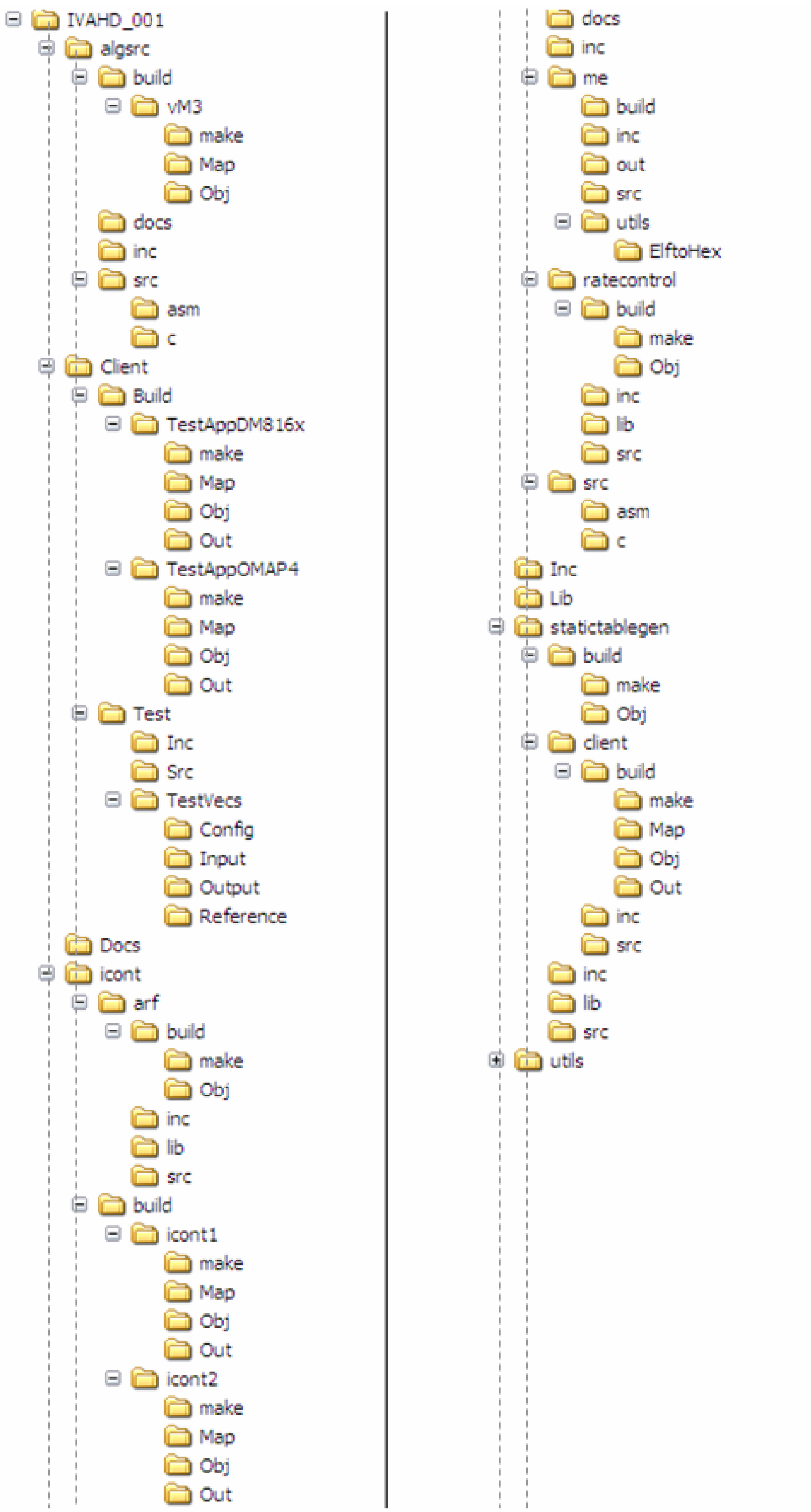


Figure 2-1. Component Directory Structure

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\Map	Contains the make file for building Media Controller lib
\algsrc\build\vM3\Map	Contains generated Map file for Media Controller (host) project
\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 the test application project. The name of this directory will not be same as exactly mentioned here. Instead of DeviceName string, actual name of Device will be present.
\Client\Build\TestAppDeviceName\Map	Contains the memory map generated on compilation of the code
\Client\Build\TestAppDeviceName\Obj	Contains the intermediate .asm and/or .obj file generated on compilation of the code
\Client\Build\TestAppDeviceName\Out	Contains the final application executable (.out) file generated by the sample test application
\Client\Test\Inc	Contains header files needed for the application code
\Client\Test\Src	Contains application C files
\Client\Test\TestVecs\Config	Contains sample configuration file for H264 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\arf\build\make	Contains the make file for building ARF lib
\icont\arf\build\obj	Contains intermediate Object files generated for ARF project
\icont\arf\inc	Contains header file related to Adaptive reference field selection module

Sub-Directory	Description
\icont\arf\lib	Contains library file related to Adaptive reference field selection module
\icont\arf\src	Contains source files needed by the Adaptive reference field selection module
\icont\build\icont1\Make	Contains the make file for building Icont 1 out file
\icont\build\icont1\Map	Contains the generated map file 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\Make\icont\build\icont2\Make	Contains the make file for building Icont 2 out file
\icont\build\icont2\Map	Contains the generated map file related to icont2 project
\icont\build\icont2\Obj	Contains the generated object files related to icont2 project
\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\inc	Contains header file related to Motion Estimation module
\icont\me\utils	Contains utility file(s) required by Motion Estimation module
\icont\ratecontrol\build\make	Contains the make file for building rate control lib
\icont\ratecontrol\build\obj	Contains intermediate Object files generated for rate control project
\icont\ratecontrol\inc	Contains header file related to Rate Control module
\icont\ratecontrol\lib	Contains library file related to Rate Control module
\icont\ratecontrol\src	Contains source file related to Rate Control 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 H.264 encoder related header files which allow interface to the codec library
\Lib	Contains the codec library file
\statictablegen\build\make	Contains the make file for building static table generation module library
\statictablegen\build\obj	Contains the generated object files for static table generation module library project

Sub-Directory	Description
\statictablegen\client\build\make	Contains the make file for building static table generation module out file
\statictablegen\client\build\map	Contains the generated map file for static table generation module out file project
\statictablegen\client\build\obj	Contains the generated object files for static table generation module out file project
\statictablegen\client\build\out	Contains the generated out file for static table generation module
\statictablegen\client\inc	Contains the header file related to static table generation module client
\statictablegen\client\src	Contains the source file related to static table generation module client
\statictablegen\inc	Contains header file related to static table generation module library
\statictablegen\lib	C Contains library file related to static table generation module
\statictablegen\src	Contains source file related to static table generation module library
\utils	Contains utility file(s) required by H.264 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) and HDVICP2 library.

This version of the codec has been validated 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 FC from following website:

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

Extract the FC zip file to the same location where you have installed Code Composer Studio. For example:

<install directory>\CCStudio4.0

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>\CCStudio4.0\<fc_directory>\packages \ti\sdo\fc\ires\hdvicp directory.
- Tiled memory related header file, these are available in the <install directory>\CStudio4.0\<fc_directory>\fctools\packages \ti\sdo\fc\ires\tiledmemory directory.
- XDM related header files, these are available in the <install directory>\CCStudio4.0\<fc_directory>\fctools\packages \ti\xdais directory.
- Memutils file for memory address translation, these are available in the <install directory>\CStudio4.0\<fc_directory>\ packages\ti\sdo\fc\memutils directory

2.3.2 Installing HDVICP2 library

The HDVICP2 library should be available in the same place as the codec package.

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

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

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

1) Verify that you have installed TI's Code Composer Studio version Version: 4.2.0.09000 and code generation tools version 4.5.1.

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

3) Verify that the following codec object libraries exist in \Lib sub-directory

   h264enc_ti_host.lib: H.264 encoder library for Ducati

4) Verify that the following codec object libraries exist in \Lib sub- directory (in case of library based / object code release):

5) h264enc_ti_icont1.out: HDVICP2.iCONT1 code

6) h264enc_ti_icont2.out: HDVICP2.iCONT2 code

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

8) Build the sample test application project by gmake

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

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

9) All files required for this project are available at the path \Client\Build\TestAppDeviceName

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

11) Select Target > Load Program on M3_Video, browse to the \Client\Build\ TestAppDeviceName\Out sub-directory, select the codec executable created in step 6, and load it into Code Composer Studio in preparation for execution.

12) 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 or EVM then this step is not needed

13) Select Target > Run on M3_Video 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.

14) On failure, the application exits with the message “Frame encoding failed”.

15) On successful completion, the application displays the information for each frame and generates output 264 bit-stream in \Client\Test\TestVecs\Output directory. User should compare with the reference provided in \Client\Test\TestVecs\Reference directory. Both the 264 bit-stream content should be same to conclude successful execution.

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.
- TestCases.txt – This file has list of config files, these needs to be executed with parameter (integer) preceding. The meaning of the parameter is below.
- 1 – execute the test case
- 0 – Terminate the regression
- For multi frame processing in single process call TestCases.txt format should be like for 4 frame processing in single process call

- 4 encoder1.cfg
- encoder2.cfg
- encoder3.cfg
- encoder4.cfg
- 0

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

# See configfile.h for a list of supported ParameterNames

################################################################################

# Files

################################################################################

InputFile =             "..\..\..\Test\TestVecs\Input\test.yuv"

EncodedFile =             "..\..\..\Test\TestVecs\Output\Test.264"

ReconFile =               "..\..\..\Test\TestVecs\Output\Test rec.yuv"

TestFile =                "..\..\..\Test\TestVecs\Reference\ref.264"

EncodingPreset = 3      # 3=> XDM USER DEFINED(see codec-specific document
                        # to understand the encoding behaviour).

RateControlPreset   = 5                 # 1 => Low Delay, 2 => Storage, 3 => Rsvd
                                        # 4 => None, 5 => user defined
```

```
MaxInterFrameInterval   = 1             # I to P frame distance. 1 indicates no B
                                     # frames. Value >1 indicates presence of
                                     # B frames.

Profile                 = 100           # Profile IDC (66=baseline, 77=main,
                                        # 100=High)

Level                   = 41            # Level IDC   (e.g. 30 = level 3.0)

NumInputUnits           = 10            # Number of units of input-data (ex. 10
                                        # Frames to be encoded).

MaxWidth                = 1920          # Max Frame width should be multiple of
                                        # 16

MaxHeight               = 1088          # Max Frame height

################################################################################

# Encoder Control

################################################################################

inputWidth             = 176     # Frame width should be multiple of 16

inputHeight            = 144     # Frame height

targetFrameRate        = 30000   # Target picture Rate per second * 1000 => For
                                 # 60 fields per second it should be 30000

targetBitRate          = 128000  # Target Bit Rate in Bits per second.

intraFrameInterval     = 10     # Interval between two consecutive intra frames,
                                # 0 => Only first frame to be intra coded, 1 =>
                                # All intra frames, N => One intra #frame and N-1
                                #  inter frames, where N > 1

interFrameInterval     = 1       # 1 - Only P frames. >1 - Number of B frames
                                 # between two I/P frames.

captureWidth           = 176     # Image width to compute image pitch. If Capture
                                 # Width is > Image Width then use the former for
                                 # image pitch.

captureHeight          = 144     # Image width to compute image pitch. If Capture
                                 # Width is > Image Width then use the former for
                                 # image pitch.
```

Any field in the `IVIDENC2_Params` structure (see Section 4.2.1.7) can be set in the Encoder.cfg file using the syntax as shown in the code snippet. If you specify additional fields in the Encoder.cfg file, ensure that you 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 conformance of the codec for other input files of your choice, follow these steps:

1) Copy the input files to the \Client\Test\TestVecs\Inputs sub-directory.

2) Copy the reference files to the \Client\Test\TestVecs\Reference sub-directory.

16) 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.

This page is intentionally left blank

Chapter 3

Sample Usage

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

3.1 Overview of the Test Application

The test application exercises the `IVIDENC2` and extended class of the H.264 Encoder library. The source files for this application are available in the \Client\Test\Src and \Client\Test\Inc sub-directories.

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 configuration file, listed in TesCases.txt and reads the various configuration parameters required for the algorithm. For more details on the configuration files, see Section 2.5.
2) Sets the interface structure based on the values it reads from the configuration file.
3) Does the algorithm instance creation and other handshake via. control methods
4) For each frame reads the input yuv frame into the application input buffer and makes a process call
5) For each frame dumps out the generated bit-stream into the specified file

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()`:

1) `algNumAlloc() - To query the algorithm about the number of memory records it requires.`
2) `algAlloc() - To query the algorithm about the memory requirement to be filled in the memory records.`
3) `algInit() - To initialize the algorithm with the memory structures provided by the application.`

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

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:

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

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

3) 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:

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

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

3) 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.

4) 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.

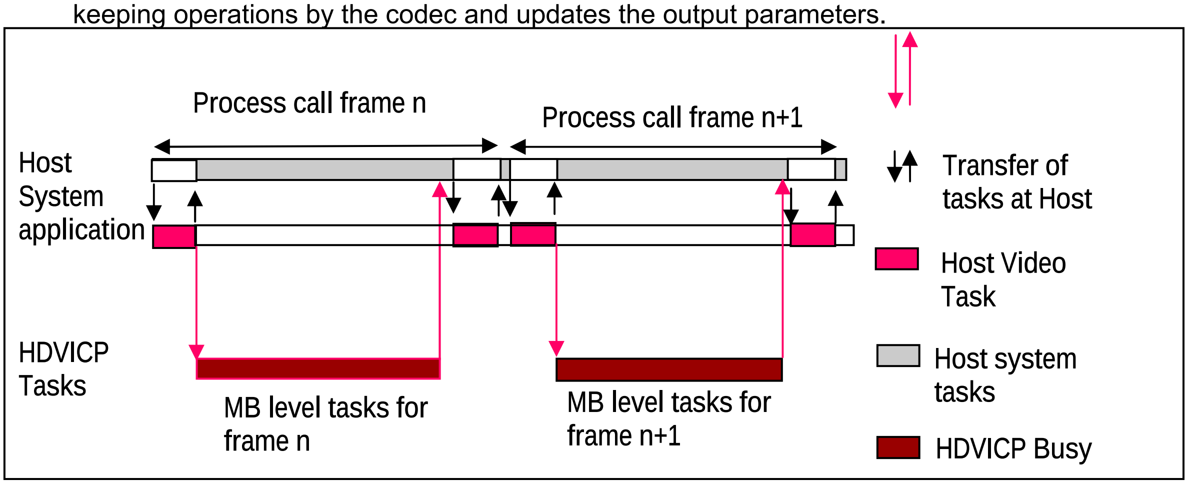


Figure 3-1. Process Call with Host Release

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:

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

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

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

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

9) 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:

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

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

3) 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 DMA and HDVICP Resource allocated for the algorithm. This is implemented by calling the RMAN interface functions in the following sequence:

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

5) RMAN_unregister() – To un-register the HDVICP protocol/resource
   manager with the generic resource manager.

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

3.2 Frame Buffer Management

3.2.1 Input Frame Buffer

The encoder has input buffers that stores frames until they are processed. These buffers at the input level are associated with a buffer input IDs. The IDs are required to track the buffers that have been processed or locked. The encoder uses this ID, at the end of the process call, to inform back to application whether it is a free buffer or not. Any buffer given to the algorithm should be considered locked by the algorithm, unless the buffer is returned to the application through `IVIDENC2_OutArgs->freeBufID[]`.For more information, see section 4.2.1.11.

For example, consider the GOP structure for IPPPP frames.

Frame Type	I	P	P	P	P
Input ID	1	2	3	4	5
Free Buffer ID	1	2	3	4	5

As shown in the table, if the input ID for the first frame is 1, the same input ID is returned as the free buffer ID at the end of the process call. There is no locking of buffers at any point.

Now, consider the GOP structure that has B frames, IBBPBBP.

Frame Type	I	B	B	P	B	B	P
Input ID	1	2	3	4	5	6	7
Free Buffer ID	0	0	1	4	2	3	7

As shown in the table, the first frame input ID (1) is returned as a free buffer ID at the end of the third process call that is after accumulating buffers for two B frames. For the first two process calls, free buffer IDs are returned as zero. This initial delay is equal to the number of B frames.

Since the 4th frame is a P frame, it is returned immediately at the end of the process call. Then, input IDs, 2 and 3 are returned as free buffers while frames 5 and 6 are being processed. Hence, if there are two B frames between P frames, the input images for the B frames are stored and the P frame is encoded first, and then the two B frames are encoded. This results in two frame period initial delay.

3.2.2 Frame Buffer Format

The frame buffer format to be used for both progressive and interlaced pictures is explained in Appendix F.

3.2.3 Address Translations

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

Application Side

process()

Codec

```
#include <…/ires_hdvicp.h>
void _MyCodecISRFunction();
MYCODEC::IVIDDEC2::process() {
  :
  …. set up for frame decode
  HDVICP_configure(h264d, h264d->hdvicpHandle,
                   H264DISRFunction);
  HDVICP_wait(h264D, h264d->hdvicpHandle);
  // Release of HOST
  …. End of frame processing
}
void H264DISRFunction(IALG_Handle handle)
{  H264D_TI_Obj *h264d = (void *)handle;

    HDVICP_done(h264d ,
                h264d->hdvicpHandle);
}
```

Framework Provided HDVICP Callback APIs

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

HDVICP_wait(handle, hdVicpHandle){


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

    SEM_post(_doneSemaphore)
}
```

Figure 3-2. 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.

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 of the same is also 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, this 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.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_BB_FRAME	Interlaced frame, both fields are B frames.
	IVIDEO_MBAFF_I_FRAME	Intra coded MBAFF frame .
	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.
IVIDENC2_Control	Process based Controls operation for Video encoder	
	IVIDENC2_CTRL_NONE IVIDENC2_CTRL_DEFAULT	No special control operation
	IVIDENC2_CTRL_FORCESKIP	Force frame to be skipped. The encoder should ignore this operation if the frame for which the control is issued is IDR/I frame.
IVIDEO_MetadataType	IVIDEO_METADATAPLANE_NONE	Used to indicate no metadata is requested or available
	IVIDEO_METADATAPLANE_MBINFO	Used to indicate that MB info metadata is requested or available
	IVIDEO_METADATAPLANE_EINFO	Used to indicate that Error info metadata is requested or available
	IVIDEO_METADATAPLANE_ALPHA	Used to indicate that Alpha metadata is requested or available
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.
	IVIDEO_INTERLACED_TOPFIELD	Interlaced video content, top field.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_INTERLACED_ BOTTOMFIELD	Interlaced video content, bottom field.
IVIDEO_RateControlPr eset	IVIDEO_LOW_DELAY	Constant Bit Rate (CBR) control for video conferencing.
	IVIDEO_STORAGE IVIDEO_RATE_CONTRO L_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_DEFINE D	User defined configuration using extended parameters.
IVIDEO_SkipMode	IVIDEO_FRAME_ENCOD ED IVIDEO_SKIPMODE_DE FAULT	Input video frame successfully encoded. Default skip mode.
	IVIDEO_FRAME_SKIPP ED	Input video frame skipped. There is no encoded bit-stream corresponding to the input frame.
IVIDEO_OutputFrameSt atus	IVIDEO_FRAME_NOERR OR IVIDEO_OUTPUTFRAME STATUS_DEFAULT	Output buffer is available (default value). Default status of the output frame.
	IVIDEO_FRAME_NOTAV AILABLE	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

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	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_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.
	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 H264 Encoder.
IVIDEO_BitRange	IVIDEO_YUVRANGE_FULL	Pixel range for YUV is 0-255.
	IVIDEO_YUVRANGE_ITU	Pixel range for YUV is as per ITU-T .
IVIDEO_DataMode	IVIDEO_FIXEDLENGTH	Data is exchanged at interval of fixed size.
	IVIDEO_SLICEMODE	Slice mode.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_NUMROWS	Number of rows, each row is 16 lines of video.
	IVIDEO_ENTIREFRAME	Processing of entire frame data.
XDM_AccessMode	XDM_ACCESSMODE_READ	Algorithm read from the buffer using the CPU.
	XDM_ACCESSMODE_WRITE	Algorithm writes to the buffer using the CPU.
XDM_CmdId	XDM_GETSTATUS	Query algorithm instance to fill Status structure.
	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 needs to initialize the dynamicParams.size and status.size fields prior to calling control() with XDM_SETDEFAULT. The algorithm must 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.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	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. The memory is not allocated by encoder and needs to be allocated by user. The buffer requirement for holding version number is of length IH264ENC_VERSION_LENGTH
	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 H264 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.
	XDM_SETLATEACQUIREARG	Set an algorithm's 'late acquire' argument. Only algorithms that utilize the late acquire IRES feature may implement this command.
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.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	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.
	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.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
XDM_MemoryUsageMode	XDM_MEMUSAGE_DATASYNC	Bit-mask to indicate the usage mode. Bit-0 is Data Sync mode. If this bit is set then it means that buffer is used in data sync mode
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. In this preset HDVICP 2.0 utilization is improved. It is supported for only high profile.interframeInterval should be 1 i.e., No B frames.
	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.
	XDM_GENERATE_HEADER	Encode only header
IVIDENC2_MotionVectorAccuracy	IVIDENC2_MOTIONVECTOR_PIXEL	Motion vectors accuracy is only integer pel.
	IVIDENC2_MOTIONVECTOR_HALFPEL	Motion vectors accuracy is half pel.
	IVIDENC2_MOTIONVECTOR_QUARTERPEL	Motion vectors accuracy is quarter pel.
	IVIDENC2_MOTIONVECTOR_EIGHTHPEL	Motion vectors accuracy is one-eighth pel.
	IVIDENC2_MOTIONVECTOR_MAX	Motion vectors accuracy is not defined.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
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 decode the bit-stream, and the decoder has concealed the bit-stream error and produced the concealed output.
	XDM_INSUFFICIENTDATA	Bit 10 ❑ 1 - Insufficient input data ❑ 0 - Ignore This error is applicable for decoders. This is set when the input data provided is not sufficient to produce 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 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 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.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_UNSUPPORTEDINPUT	Bit 13 ❑ 1 – Un-supported 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.
	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 encoder does not support sliceMode for below 128x80 resolution, it will return XDM_UNSUPPORTEDPARAM when the control function is called for parameter validation.
	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: Used for codec specific error codes.
- Bit 0-7: Codec and implementation specific (see Table 4-4)

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

Table 4-2. H264 Encoder Specific Enumerated Data Types.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IH264ENC_Intra4x4Params	H.264 Encoder slice level control for Intra4x4 modes	
	IH264_INTRA4x4_NONE	Disable Intra4x4 modes
	IH264_INTRA4x4_ISLICES	Enable Intra4x4 modes only in I Slices
	IH264_INTRA4x4_IPBSLICES IH264_INTRA4x4_DEFAULT	Enable Intra4x4 modes only in I, P and B Slices. This is the default setting.
IH264ENC_Level	IH264_LEVEL_10	H.264 Level 1.0
	IH264_LEVEL_1b	H.264 Level 1.b
	IH264_LEVEL_11	H.264 Level 1.1
	IH264_LEVEL_12	H.264 Level 1.2
	IH264_LEVEL_13	H.264 Level 1.3
	IH264_LEVEL_20	H.264 Level 2.0
	IH264_LEVEL_21	H.264 Level 2.1
	IH264_LEVEL_22	H.264 Level 2.2
	IH264_LEVEL_30	H.264 Level 3.0
	IH264_LEVEL_31	H.264 Level 3.1
	IH264_LEVEL_32	H.264 Level 3.2
	IH264_LEVEL_40	H.264 Level 4.0
	IH264_LEVEL_41	H.264 Level 4.1
	IH264_LEVEL_42	H.264 Level 4.2
	IH264_LEVEL_50	H.264 Level 5.0
	IH264_LEVEL_51	H.264 Level 5.1
IH264ENC_Profile	Profile identifier for H.264 Encoder	
	IH264_BASELINE_PROFILE	Baseline profile
	IH264_MAIN_PROFILE	Main profile
	IH264_EXTENDED_PROFILE	Extended profile
	IH264_HIGH_PROFILE IH264_DEFAULT_PROFILE	High profile. This is the default setting.
	IH264_HIGH10_PROFILE	High 10 profile

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264_HIGH422_PROFILE	High 4:2:2 profile
IH264ENC_MetadataType	Meta data type specifc to H.264 encoder	
	IH264_SEI_USER_DATA_UNREGISTERED	H.264 allows inserting SEI message for any user data, refer section D.1.6 of H.264 standard. By setting this value to any of IVIDENC2_Params::metadataType[i] You can provide the data SEI to be inserted in H.264 bit-stream Refer IH264ENC_MetaDataFormatUserDefinedSEI for the format of user data.
	IH264_REGION_OF_INTEREST	By setting this value to any of IVIDENC2_Params::metadataType[i] You can provide region of interest information for smart encoding. This is not supported in H.264 Encoder 2.0
	IH264_USER_DEFINED_SCALINGMATRIX	By setting this value to any of IVIDENC2_Params::metadataType[i] You can provide scaling matrices to be used by encoder. Refer Appendix C for more details.
IH264ENC_LTRPScheme	IH264ENC_LTRP_NONE	No longterm refernce frame
	IH264ENC_LTRP_REFERTO_PERIODICLTRP	Mark frames as long-term reference frame with the period given by LTRPPeriod of IH264ENC_Params and based on the frame control IH264ENC_Control
	IH264ENC_LTRP_REFERTOP_PROACTIVE	Two long term frames are supported in this scheme and long-term index marking and refernce frame update is done based the IH264ENC_Control values
	IH264ENC_LTRP_REFERTOP_REACTIVE	Mark frames as long-term reference frame with the period given by LTRPPeriod of IH264ENC_Params. At any point of time there will be 2 long-term frames and based on the frame control IH264ENC_Control

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IH264ENC_Control	Picture level control	
	IH264ENC_CTRL_REFER_LONG_TERM_FRAME	Control to encoder for referring long term reference frame
	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	Control to encoder for encoding current frame as non-referencing P frame and not to update reference frame for this P frame
	IH264ENC_CTRL_WRITE_NOREFUPDATE	Control to encoder for encoding current frame as referencing P frame and not to update reference frame for this P frame
	IH264ENC_CTRL_NOWRITE_REFUPDATE	Control to encoder for encoding current frame as non-referencing P frame and to update reference frame for this P frame
	IH264ENC_CTRL_WRITE_REFUPDATE	Control to encoder for encoding current frame as referencing P frame and to update reference frame for this P frame
	IH264ENC_CTRL_START_GDR	Control to start GDR activity. Applicable when intraRefreshMethod is IH264_INTRAREFRESH_GDR
IH264ENC_PicOrderCountType	Picture Order Count Type Identifier	
	IH264_POC_TYPE_0 IH264_POC_TYPE_DEFAULT	POC type 0. Default POC type to be used by encoder.
	IH264_POC_TYPE_1	POC type 1
	IH264_POC_TYPE_2	POC type 2
IH264ENC_ScalingMatPreset	Controls the type of scaling matrix picked up by encoder	
	IH264_SCALINGMATRIX_NONE	Flat scaling matrix: part of standard (no scaling matrix). Default (if profile != HIGH)
	IH264_SCALINGMATRIX_NORMAL IH264_SCALINGMATRIX_STD_DEFAULT	For normal contents (normal contents). Default (if profile == HIGH)
	IH264_SCALINGMATRIX_NOISY	For noisy contents.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264_SCALINGMATRIX_USERDEFINED_SPSLEVEL	Scaling matrices can be provided at SPS level. See Appendix C for more details
	IH264_SCALINGMATRIX_USERDEFINED_SPSLEVEL	Scaling matrices can be provided by at PPS level. See Appendix C for more details
IH264ENC_RateControlAlgo	These enumerations control the type of rate control algorithm to be picked up by the encoder. Only useful if IVIDENC2::rateControlPreset is set as IVIDEO_USER_DEFINED.	
	IH264_RATECONTROL_PRC IH264_RATECONTROL_DEFAULT	Perceptual Rate Control, controls the QP at MB level with VBR mode Default rate control algorithm.
	IH264_RATECONTROL_PRC_LOW_DELAY	Perceptual Rate Control, controls the QP at MB level with CBR (Low delay) mode
IH264ENC_FrameQualityFactor	These enumerations control the quality factor between two types of frames, I frame quality with respect to P frame. For example, higher quality factor means I frame quality is given higher importance compared to P frame.	
	IH264_QUALITY_FACTOR_1 IH264_QUALITY_FACTOR_DEFAULT	Same quality factor between two types of frame. It is default quality factor.
	IH264_QUALITY_FACTOR_2	High quality factor to one frame type between two types of frame.
	IH264_QUALITY_FACTOR_3	Higher quality factor to one frame type between two types of frame.
IH264ENC_RateControlParamsPreset	These enumerations control the rate control parameters. This preset controls the USER_DEFINED versus DEFAULT mode. If you are not aware about the following fields, it should be set as IH264_RATECONTROLPARAMS_DEFAULT.	
	IH264_RATECONTROLPARAMS_DEFAULT	Default rate control params.
	IH264_RATECONTROLPARAMS_USERDEFINED	User defined rate control params. Default value.
	IH264_RATECONTROLPARAMS_EXISTING	Keep the rate control params as existing. This is useful during control call, if user does not want to change the rate control parameters.
IH264ENC InterCodin	These enumerations control the type of inter coding.	

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
gPreset	IH264_INTERCODING_DEFAULT	Default inter coding params.
	IH264_INTERCODING_USERDEFINED	User defined inter coding params. Default value.
	IH264_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.
	IH264_INTERCODING_MED_SPEED_HIGH_QUALITY	InterCoding Preset for Medium speed high quality encoding
	IH264_INTERCODING_HIGH_SPEED	InterCoding Preset for High speed encoding. This is supported only when all the below conditions (a,b,c) are satisfied a.enablePartialFrameSkip should be disabled. b.intraRefreshMethod should be default. c.transformBlockSize should be IH264_TRANSFORM_8x8.
IH264ENC_MeAlgoMode	IH264ENC_MOTIONESTMODE_NORMAL IH264ENC_MOTIONESTMODE_DEFAULT	Motion estimation algorithm selection for normal encoding
	IH264ENC_MOTIONESTMODE_HIGH_SPEED	Motion estimation algorithm selection for high speed encoding. This is supported only when all the below conditions (a,b,c) are satisfied. a)IVIDENC2_DynamicParams::interFrameInterval is '1' b) IVIDENC2_DynamicParams :: mvAccuracy == IVIDENC2_MOTIONVECTOR_QUARTERPEL c)IH264ENC_InterCodingParams :: minBlockSizeP == IH264_BLOCKSIZE_16x16
IH264ENC_IntraCodingBias	IH264ENC_INTRACODINGBIAS_NORMAL IH264ENC_INTRACODINGBIAS_DEFAULT	IntraCoding Bias for normal encoding. No special restriction on number of intra macro blocks.
	IH264ENC_INTRACODINGBIAS_HIGH_SPEED	Puts special restriction on intra macro blocks to limit it to 12 % of total Mbs in the picture.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IH264ENC_InterBlockSize	These enumerations are defined for minimum inter block size.	
	IH264_BLOCKSIZE_16x16 IH264_BLOCKSIZE_DEFAULT	16x16 block size. It is also default block size.
	IH264_BLOCKSIZE_8x8	8x8 block size
	IH264_BLOCKSIZE_4x4	4x4 Block size Not supported in this version of H264 Encoder
IH264ENC_BiasFactor	Control to code the macro block as inter or intra. Also, for having a macro block use skip MV or regular MV.	
	IH264_BIASFACTOR_LOW	Low biasing.
	IH264_BIASFACTOR_MEDIUM IH264_BIASFACTOR_NORMAL IH264_BIASFACTOR_DEFAULT	Normal/Med biasing. Default biasing factor.
	IH264_BIASFACTOR_MILD	Mild bias factor
	IH264_BIASFACTOR_ADAPTIVE	Adaptive bias factor
	IH264_BIASFACTOR_HIGH	High biasing.
IH264ENC_IntraRefreshMethods	Refresh method type identifier for H.264 Encoder.	
	IH264_INTRAREFRESH_NONE IH264_INTRAREFRESH_DEFAULT	Does not forcefully insert intra macro blocks. Default intra refresh is OFF.
	IH264_INTRAREFRESH_CYCLIC_MBS	Inserts intra macro blocks in a cyclic mode. Cyclic interval is equal to intraRefreshRate.
	IH264_INTRAREFRESH_CYCLIC_SLICES	Inserts intra slices (row based) in a cyclic mode: Cyclic interval is equal to intraRefreshRate.
	IH264_INTRAREFRESH_RDOPT_MBS	Position of intra macro blocks is chosen by encoder, but the number of forcefully coded intra macro blocks in a frame is guaranteed to be equal to totalMbsInFrame/intraRefreshRate.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264_INTRAREFRESH_GDR	Instead of a sudden Intra refresh of entire frame, the frame is refreshed gradually over a duration (which is con figerable) of frames with refresh happening by Intra coded rows scanning from top to bottom of the scene/picture This method of intra refresh mechanism is not supported for interlaced cases and in case of 'B'frames i.e., IVIDENC2_Params::inputContentType should not be IVIDEO_INTERLACED **(no interlace)** and interFrameInterval == 1 (no 'B' frames)
IH264ENC_ChormaComponent	These enumerations control the selection of chroma component to perform chroma intra estimation.	
	IH264_CHROMA_COMPONENT_CR_ONLY IH264_CHROMA_COMPONENT_DEFAULT	Only Cr component Default is Only CR component.
	IH264_CHROMA_COMPONENT_CB_CR_BOTH	Both Cb and Cr component.
IH264ENC_IntraCodingPreset	These enumerations control the type of intra coding.	
	IH264_INTRACODING_DEFAULT	Default intra coding params.
	IH264_INTRACODING_USERDEFINED	User defined intra coding params. Default value.
	IH264_INTRACODING_EXISTING	Keep intra coding params as existing. This is useful during control call, if you do not want to change the inter coding params
	IH264_INTRACODING_HIGH_SPEED	Intra coding params for high speed encoding. Me algo mode should be IH264ENC_MOTIONESTMODE_HIGH_SPEED. IH264ENC_TransformBlockSize should be IH264_TRANSFORM_8x8.
IH264ENC_NALUnitType	IH264_NALU_TYPE_SPS_WITH_VUI	Sequence parameter set having VUI information.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264_NALU_TYPE_SLICE	Slice of a non-IDR picture.
	IH264_NALU_TYPE_SLICE_DP_A	Coded slice data partition A.
	IH264_NALU_TYPE_SLICE_DP_B	Coded slice data partition B.
	IH264_NALU_TYPE_SLICE_DP_C	Coded slice data partition C.
	IH264_NALU_TYPE_IDR_SLICE	Slice of an IDR picture.
	IH264_NALU_TYPE_SEI	Supplemental enhancement information.
	IH264_NALU_TYPE_SPS	Sequence parameter set.
	IH264_NALU_TYPE_PPS	Picture parameter set.
	IH264_NALU_TYPE_AUD	Access unit delimiter.
	IH264_NALU_TYPE_EOSEQ	End of sequence.
	IH264_NALU_TYPE_EOSTREAM	End of stream.
	IH264_NALU_TYPE_FILLER	Filler data.
	IH264_NALU_TYPE_SPS_EXT	Sequence parameter set extension.
	IH264_NALU_TYPE_USER_DATA_UNREGD_SEI	User data un-registered SEI.
IH264ENC_NALUControlPreset	These enumerations define the control mechanism for insertion of different NALU types at different point in video sequence.	
	IH264_NALU_CONTROL_DEFAULT	Default NALU insertion.
	IH264_NALU_CONTROL_USERDEFINED	User defined NALU insertion.
IH264ENC_SliceCodingPreset	These enumerations control the type of slice coding.	
	IH264_SLICECODING_DEFAULT	Default slice coding params.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264_SLICECODING_USERDEFINED	User defined slice coding params. Default value.
	IH264_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.
IH264ENC_SliceMode	These enumerations control the mode of slice coding.	
	IH264_SLICEMODE_NONE IH264_SLICEMODE_DEFAULT	Single Slice per picture. This is the default slice coding mode.
	IH264_SLICEMODE_MBUNIT	Slices are controlled based upon number of macro blocks.
	IH264_SLICEMODE_BYTES	Slices are controlled based on number of bytes.
	IH264_SLICEMODE_OFFSET	Slices are controlled based on user defined offset in unit of rows.
IH264ENC_StreamFormat	These enumerations control the type stream format.	
	IH264_BYTE_STREAM IH264_STREAM_FORMAT_DEFAULT	Bit-stream contains the start code identifier. Default slice coding mode.
	IH264_NALU_STREAM	Bit-stream does not contain the start code identifier.
IH264ENC_LoopFilterPreset	Controls the loop filter preset options	
	IH264_LOOPFILTER_DEFAULT	Default loop-filtering params.
	IH264_LOOPFILTER_USERDEFINED	User defined loop-filtering params.
IH264ENC_LoopFilterDisableIDC	Controls H264 loop filter disable options	
	IH264_DISABLE_FILTER_NONE IH264_DISABLE_FILTER_DEFAULT	Enable filtering of all the edges. Default is loop filter enabled.
	IH264_DISABLE_FILTER_ALL_EDGES	Disable filtering of all the edges.
	IH264_DISABLE_FILTER_SLICE_EDGES	Disable filtering of slice edges.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IH264ENC_SliceGroupMapType	Map type of slice group.	
	IH264_INTERLEAVED_SLICE_GRP	Interleaved slice group.
	IH264_DISPERSED_SLICE_GRP IH264_SLICE_GRP_MAP_DEFAULT	Dispersed slice group. Default value.
	IH264_FOREGRND_WITH_LEFTOVER_SLICE_GRP	ForeGround with Left Over.
	IH264_BOX_OUT_SLICE_GRP	Box Out.
	IH264_RASTER_SCAN_SLICE_GRP	Raster Scan.
	IH264_WIPE_SLICE_GRP	Wipe slice group.
	IH264_EXPLICIT_SLICE_GRP	Explicit Slice group map type.
IH264ENC_SliceGroupChangeDirection	Only valid when sliceGroupMapType is equal to IH264_RASTER_SCAN_SLICE_GRP, IH264_WIPE_SLICE_GRP, or IH264_WIPE_SLICE_GRP.	
	IH264_RASTER_SCAN IH264ENC_SLICEGROUP_CHANGE_DIRECTION_DEFAULT	Raster scan order. Default slice group direction.
	IH264_CLOCKWISE	Clockwise (used for box out FMO parameters).
	IH264_RIGHT	Right, used for Wipe FMO type.
	IH264_REVERSE_RASTER_SCAN	Reverse raster scan order.
	IH264_COUNTER_CLOCKWISE	Counter clockwise, used for box out FMO parameters.
	IH264_LEFT	Left, used for Wipe FMO type.
IH264ENC_FMOCodingPreset	Controls for FMO coding preset	
	IH264_FMOCODING_NONE IH264_FMOCODING_DEFAULT	No FMO Default FMO coding value
	IH264_FMOCODING_USERDEFINED	User defined FMO parameters

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IH264ENC_EntropyCodingMode	Controls the entropy coding type	
	IH264_ENTROPYCODING_CAVLC IH264_ENTROPYCODING_DEFAULT	CAVLC coding type
	IH264_ENTROPYCODING_CABAC	CABAC coding type
IH264ENC_TransformBlockSize	In H264 intra macro block, transform size depends on the intra mode, so this applies to inter macro blocks only.	
	IH264_TRANSFORM_4x4	Transform blocks size is 4x4
	IH264_TRANSFORM_8x8	Transform blocks size is 8x8 : Valid for only High Profile
	IH264_TRANSFORM_ADAPTIVE IH264_TRANSFORM_DEFAULT	Adaptive transform block size: encoder decides as per content
IH264ENC_GOPStructure	Type of Group of Pictures (GOP)	
	IH264ENC_GOPSTRUCTURE_NONUNIFORM IH264ENC_GOPSTRUCTURE_DEFAULT	Open GOP structure: IBBPBBP Default
	IH264ENC_GOPSTRUCTURE_UNIFORM	Close GOP structure: BBIBBPBB
IH264ENC_InterlaceCodingType	Controls the type of interlaced coding	
	IH264_INTERLACE_PICAFF	PicAFF type of interlace coding
	IH264_INTERLACE_MBAFF	MBAFF type of interlace coding
	IH264_INTERLACE_FIELDONLY IH264_INTERLACE_FIELDONLY_MRF	Field only coding with selecting most recent field as reference
	IH264_INTERLACE_FIELDONLY_ARF IH264_INTERLACE_DEFAULT	Field only coding where codec decides the parity of the field to be used based on content. Default setting
	IH264_INTERLACE_FIELDONLY_SPF	Field only coding with selecting same parity field as reference.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IH264ENC_VUICodingPreset	Preset for VUI related parameters	
	IH264_VUICODING_DEFAULT	Default VUI Parameters. Note that Enable/Disable of VUI is through nalUnitControlParams
	IH264_VUICODING_USERDEFINED	User defined VUI parameters
IH264ENC_StereoInfoPreset	Preset for StereoInfo parameters	
	IH264_STEREOINFO_DISABLE	Disable Stereo Video Coding.
	IH264_STEREOINFO_ENABLE_DEFAULT	Enable stereo video coding in default mode
	IH264_STEREOINFO_ENABLE_USERDEFINED	Enable stereo video coding in userdefined mode.
IH264ENC_FramePackingPreset	Preset for Frame packing SEI parameters	
	IH264_FRAMEPACK_SEI_DISABLE	Disable frame packing SEI.
	IH264_FRAMEPACK_SEI_ENABLE_DEFAULT	Enable frame packing SEI coding in default mode
	IH264_FRAMEPACK_SEI_USERDEFINED	Enable frame packing SEI coding in userdefined mode.
IH264ENC_FramePackingType	Enumerations for Frame Packing arrangement type	
	IH264_FRAMEPACK_CHECKERBOARD	Checker board arrangement of 2 views
	IH264_FRAMEPACK_COLUMN_INTERLEAVING	Column interleaving arrangement of 2 views
	IH264_FRAMEPACK_ROW_INTERLEAVING	Row interleaving arrangement of 2 views
	IH264_FRAMEPACK_SIDE_BY_SIDE IH264_FRAMEPACK_TYPE_DEFAULT	Side by side arrangement of 2 views
	IH264_FRAMEPACK_TOP_BOTTOM	Top-Bottom arrangement of 2 views
IH264ENC_VideoFormat	Video format for VUI parameters	

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264ENC_VIDEOFORMAT_COMPONENT	Component video format
	IH264ENC_VIDEOFORMAT_PAL	PAL video format
	IH264ENC_VIDEOFORMAT_NTSC	NTSC video format
	IH264ENC_VIDEOFORMAT_SECAM	SECAM video format
	IH264ENC_VIDEOFORMAT_MAC	MAC video format
	IH264ENC_VIDEOFORMAT_UNSPECIFIED	Unspecified video format
IH264ENC_AspectRatioIdc	Enumeration for aspect ratio	
	IH264ENC_ASPECTRATIO_UNSPECIFIED	Unspecified aspect ratio
	IH264ENC_ASPECTRATIO_SQUARE	1:1 (square) aspect ratio
	IH264ENC_ASPECTRATIO_12_11	12:11 aspect ratio
	IH264ENC_ASPECTRATIO_10_11	10:11 aspect ratio
	IH264ENC_ASPECTRATIO_16_11	16:11 aspect ratio
	IH264ENC_ASPECTRATIO_40_33	40:33 aspect ratio
	IH264ENC_ASPECTRATIO_24_11	24:11 aspect ratio
	IH264ENC_ASPECTRATIO_20_11	20:11 aspect ratio
	IH264ENC_ASPECTRATIO_32_11	32:11 aspect ratio
	IH264ENC_ASPECTRATIO_80_33	80:33 aspect ratio
	IH264ENC_ASPECTRATIO_18_11	18:11 aspect ratio
	IH264ENC_ASPECTRATIO_15_15	15:15 aspect ratio

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264ENC_ASPECTRATIO_64_33	64:33 aspect ratio
	IH264ENC_ASPECTRATIO_160_99	160:99 aspect ratio
	IH264ENC_ASPECTRATIO_4_3	4:3 aspect ratio
	IH264ENC_ASPECTRATIO_3_2	3:2 aspect ratio
	IH264ENC_ASPECTRATIO_2_1	2:1 aspect ratio
	IH264ENC_ASPECTRATIO_EXTENDED	Extended aspect ratio
IH264ENC_RoiType	Enumeration for different ROI types	
	IH264_FACE_OBJECT	ROI is of FACE_OBJECT type
	IH264_BACKGROUND_OBJECT	ROI is of BACKGROUND_OBJECT type
	IH264_FOREGROUND_OBJECT	ROI is of FOREGROUND_OBJECT type
	IH264_DEFAULT_OBJECT	ROI is of DEFAULT_OBJECT type
	IH264_PRIVACY_MASK	ROI is of PRIVACY_MASK type
IH264ENC_NumTemporalLayer	IH264_TEMPORAL_LAYERS_1	Only base layer
	IH264_TEMPORAL_LAYERS_2	Base layer + Temporal layer
	IH264_TEMPORAL_LAYERS_3	Base layer + 2Temporal layers
	IH264_TEMPORAL_LAYERS_4	Base layer + 3Temporal layers
	IH264_TEMPORAL_LAYERS_MAX	Maximum temporal layer supported

Table 4-3. H264 Encoder Constants

Constant Name	Value	Description of Constant

Constant Name	Value	Description of Constant
IVIDENC2_DEFAULTPROFILE	-1	This constant is used when a particular codec doesn't have a profile, or the application doesn't know which profile the codec should use.
IVIDENC2_DEFAULTPLEVEL	-1	This constant is used when a particular codec doesn't have a level, or the application doesn't know which profile the codec should use.
IH264ENC_MAXNUMSLCGPS	2	Maximum number of slice groups.
IH264ENC_VERSION_LENGTH	64	Length of the version string. The memory to get version number is owned by application.
IH264ENC_MAX_NUM_SLICE_START_OFFSET	3	Maximum Number of slice start points.
IH264ENC_MAX_SEI_METADTA_BUFSIZE	0x3FF	Maximum size for SEI_USER_DATA_UNREGISTERED SEI message.
IH264ENC_MAX_ROI	36	Maximum number of ROI rectangles

Table 4-4. H.264 Encoder Error Statuses

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IH264ENC_ErrorBit	IH264ENC_LEVEL_INCOMPLAINT_PARAMETER	Bit 0 - level non-compliant parameters. This error is applicable when some parameters are set, which are not meeting the limit defined by H.264 standard Table A-1 Level limits. The error can be categorized under following category : ❑ IH264ENC_LEVEL_INCOMPLAINT_RESOLUTION : Invalid width/height ❑ IH264ENC_LEVEL_INCOMPLAINT_HRDBUFSZIE : Invalid HrdBufferSize ❑ IH264ENC_LEVEL_INCOMPLAINT_BITRATE : Invalid Bit Rate ❑ IH264ENC_LEVEL_INCOMPLAINT_MBSPERSECOND : Invalid FrameRate/resolution ❑ IH264ENC_LEVEL_INCOMPLAINT_DPBSIZE : Invalid DPB size For above 5 situations, only a signal bit (bit-0) is set as true

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264ENC_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 IH264_BASELINE_PROFILE.
	IH264ENC_PROFILE_INCOMPLAINT_FMO_SETTING	Bit 2 - Profile in-complaint FMO setting. This error is applicable when FMO is enabled but IVIDENC2_Params::profile is not set as IH264_BASELINE_PROFILE.
	IH264ENC_PROFILE_INCOMPLAINT_TRANSFORMBLOCKSIZE	Bit 3 - Profile in-complaint transform block size. This error is set when IH264ENC_Params::transformBlockSize != IH264_TRANSFORM_4x4 && IVIDENC2_Params::profile != IH264_HIGH_PROFILE.
	IH264ENC_PROFILE_INCOMPLAINT_INTERFRAMEINTERVAL	Bit 4 - Profile in-complaint, inter frame interval. This error is set when B frames are used with IH264_BASELINE_PROFILE.
	IH264ENC_PROFILE_INCOMPLAINT_SCALINGMATRIXPRESET	Bit 5 - Profile in-complaint scaling matrix setting. This error is set when scaling matrix is used without IH264_HIGH_PROFILE.
	IH264ENC_PROFILE_INCOMPLAINT_ENTROPYCODINGMODE	Bit 6 - Profile in-complaint entropy coding mode setting. This error is set when cabac is used without IH264_HIGH_PROFILE/MAIN_PROFILE. This is create time error
	IH264ENC_MAX_BYTES_VOILATION_IN_SLICEMODE_BYTES	Bit 6 - If number of bytes encoded in any of the slice in the currently encoded picture is crossing maximum unit size then this bit will be set. This is run time error produced during encoding of a frame This error bit is shared with IH264ENC_PROFILE_INCOMPLAINT_ENTROPYCODINGMODE.Both Erroneous situations are mutually exclusive hence the bits are shared

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264ENC_MAX_BIT_RATE_VOILATION	Bit 7 – Max bit rate violation Under some situations, encoder might not be able to meet max bit rate. This bit is set when bits consumed in one unit (1 sec) is more than the allocated as per the given max bit rate. If the frame rate is N , and if the max bit rate is violated in Mth frame than this bit will get set for frame M to N. (M <= N)
	IH264ENC_IMPROPER_HDVICP2_STATE	Bit 16 – HDVCIP2 is not in proper state, before using the HDVICP2, 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
	IH264ENC_IMPROPER_STREAMFORMAT	Bit 17 - Stream format is not proper. This error is set when streamFormat is set as IH264_NALU_STREAM but data synch is not enabled for put data.
	IH264ENC_IMPROPER_POCTYPE	Bit 18 - POC type is not proper. This error is set when POC type 2 is used in presence of non reference frames.
	IH264ENC_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.
	IH264ENC_UNSUPPORTED_VIDENC2PARAMS	Bit 20 - Invalid videnc2 parameters. This error is set when any parameter of structure IVIDENC2_Params is not in allowed range.
	IH264ENC_UNSUPPORTED_RATECONTROLPARAMS	Bit 21 - Invalid rate control parameters. This error is set when any parameter of structure IH264ENC_RateControlParams is not in allowed range.
	IH264ENC_UNSUPPORTED_INTERCODINGPARAMS	Bit 22 - Invalid inter coding parameters. This error is set when any parameter of structure IH264ENC_InterCodingParams is not in allowed range.
	IH264ENC_UNSUPPORTED_INTRACODINGPARAMS	Bit 23 - Invalid Intra coding parameters. This error is set when any parameter of structure IH264ENC_IntraCodingParams is not in allowed range.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264ENC_UNSUPPORTED_NALUNITCONTROLPARAMS	Bit 24 - Invalid NAL unit coding parameters. This error is set when any parameter of structure IH264ENC_NALUControlParams is not in allowed range.
	IH264ENC_UNSUPPORTED_SLICECODINGPARAMS	Bit 25 - Invalid slice coding parameters This error is set when any parameter of structure IH264ENC_SliceCodingParams is not in allowed range
	IH264ENC_UNSUPPORTED_LOOPFILTERPARAMS	Bit 26 - Invalid loop filter related parameters This error is set when any parameter of structure IH264ENC_LoopFilterParams is not in allowed range
	IH264ENC_UNSUPPORTED_FMOCODINGPARAMS	Bit 27 - Invalid FMO parameters This error is set when any parameter of structure IH264ENC_FMOCodingParams is not in allowed range
	IH264ENC_UNSUPPORTED_N_FRAME_PROCESSCALL_PARAMS	Bit 27 – This error bit is set when unsupported parameter for N frame process call is provided to codec.
	IH264ENC_DATASYNCH_RUN_TIME_ERROR	Bit 27 – Error bit to indicate run time data synch errors mentioned below ❑ when number of NALs in 1KB of data is more than 64 This error bit is shared with IH264ENC_UNSUPPORTED_FMOCODINGPARAMS. Both Erroneous situations are mutually exclusive hence the bits are shared
	IH264ENC_UNSUPPORTED_VUICODxINGPARAMS	Bit 28 - Invalid VUI coding parameters This error is set when any parameter of structure IH264ENC_VUICodingParams is not in allowed range
	IH264ENC_UNSUPPORTED_H264ENCPARAMS	Bit 29 - Invalid Create time extended parameters This error is set when any parameter of structure IH264ENC_Params is not in allowed range
	IH264ENC_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

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264ENC_UNSUPPORTED_H264ENCDYNAMICPARAMS	Bit 31 - Invalid extended class dynamic parameters during control This error is set when any parameter of structure IH264ENC_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_Int16`	Input	Type of memory, See `XDM_MemoryType` enumeration in Table 4-1 for more details
`usageMode`	`XDAS_Int16`	Input	Memory usage descriptor, this field is set by the owner of the buffer (typically the application), and read by users of the buffer (including the algorithm). See `XDM_MemoryUsageMode` enumeration 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 (for 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.

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 H.264 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 (Supported `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.
- When the input frame buffer that getting encoded by encoder is not same as capture buffer then encoder still returns the size of

the buffer accessed by him. In these situations application should take care of proper buffer allocation for input frame buffer

These are the example buffer sizes but you can re-configure 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 4-1 for more details
planeDesc [IVIDEO_MAX_NUM_PLANES]	XDM2_SingleBufDesc	Input/Output	Description for video planes
metadataPlaneDesc [IVIDEO_MAX_NUM_METADATA_PLA	XDM2_SingleBufDesc	Input/Output	Description for metadata planes

Field	Data Type	Input/ Output	Description
NES]			
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[IVIDEO_MAX_NUM_PLANES]	XDAS_Int32	Input/Output	Image pitch for each plane
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. Not applicable for encoder input buffers.
repeatFrame	XDAS_Int32	Input/Output	Number of times the display process

Field	Data Type	Input/ Output	Description
			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. 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 4-1 shows `IVIDEO2_BufDesc` structure with the associated variables.

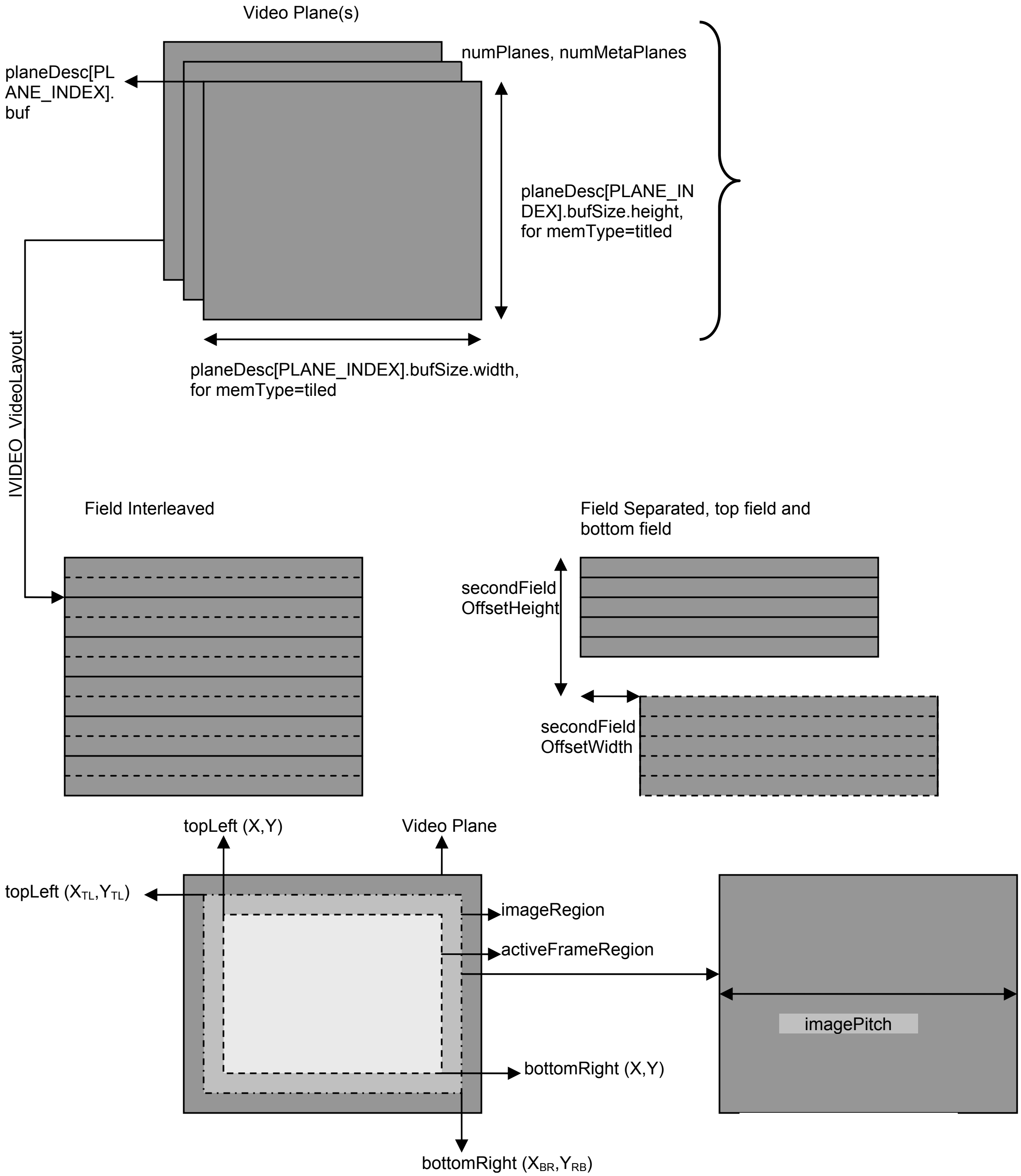


Figure 4-1. IVIDEO2_BufDesc With Associated Parameters.

> Note:
>
> The following table provides the number of process calls that needs to be made for interlaced versus progressive for different cases.
>
> | ID | content Type | input Width | input Height | target FrameRate | dataLayout | secondField OffsetWidth/ Height |
> |---|---|---|---|---|---|---|
> | 0 | IVIDEO_PROGRESSIVE | 1920 | 1088 | 30000 | Ignore | Ignore |
> | 1 | IVIDEO_INTERLACED | 1920 | 544 | 30000 | IVIDEO_FIELD_SEPARATED | Non zero |
> | 2 | IVIDEO_INTERLACED | 1920 | 544 | 30000 | IVIDEO_FIELD_INTERLEAVED | Ignore |
> | 3 | IVIDEO_INTERLACED | 1920 | 544 | 30000 | IVIDEO_FIELD_SEPARATED | 0,0 |
>
> **0**: 1920x1080p requires 30 process calls
>
> **1**: 1920x1080i requires 30 process calls, where each call accepts two fields in field separated format
>
> **2**: 1920x1080i requires 30 process calls, where each call accepts two fields in field interleaved format
>
> **3**: 1920x1080i requires 60 process calls, where each call accepts one field
>
> - Co-ordinates of imageRegion and activeFrameRegion should not be –ve. There is no error check perfromed by encoder for this case
> - bufSize structure of planeDesc doesn't carry any meaning. Buffer size is assumed to be sufficient as per width and height, so it is don't care
> - imagePitch is don't care if the memType != PAGE and RAW
> - In other cases imagePitch = 0 means same as width and other values of imagePitch are valid and user responsibility to provide correct value

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 4-5.

|| 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(IH264ENC_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.
maxWidth	XDAS_Int32	Input	Maximum video width to be supported in pixels.
dataEndianness	XDAS_Int32	Input	Endianness of output data. See `XDM_DataFormat` enumeration in Table 4-1 for more details.

Field	Data Type	Input/ Output	Description
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: ❑ 1 - No B-frames ❑ 2 - Insert one B-frame. ❑ 3 - Insert two B frames ❑ N - Insert N-1 B frames between two P frames.
maxBitRate	XDAS_Int32	Input	This parameter along with the IVIDENC2_DynamicParams :: targetBitRate is used to **Enable/Disable High Fidelity Variable Bitrate (HFVBR/CVBR)** Rate Control. Refer Appendix N HFVBR for more details.
minBitRate	XDAS_Int32	Input	Minimum bit rate for encoding in bits per second
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.
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. See IH264ENC_Profile enumeration in Table 4-2 for more details.
Level	XDAS_Int32	Input	Level indicator of video encoder. See IH264ENC_Level enumeration in Table 4-2 for details.
inputDataMode	XDAS_Int32	Input	Input data mode. See IVIDEO_DataMode enumeration in Table 4-1 for details.
outputDataMode	XDAS_Int32	Input	Output data mode. See IVIDEO_DataMode enumeration in Table 4-1 for details.
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.

Field	Data Type	Input/ Output	Description
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.
metadataType[IVIDEO_MAX_NUM_METADATA_PLANES]	XDAS_Int32	Input	Type of the each meta data plane, refer IVIDEO_MetadataType (or extended enumeration) for possible values

Note:

The following fields of IVIDENC2_Params data structure are level dependent:

- maxHeight
- maxWidth
- maxInterFrameInterval

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 14496-10* for the supported maxFrameSizeinMbs values.

For example, consider you have to check if the following values are supported for level 2.0:

- maxHeight = 480
- maxWidth = 720

The supported maxFrameSizeinMbs value for level 2.0 as per Table A.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 MaxDPB size value by referring to Table A.1 – Level Limits and make sure currDPBsize <= MaxDPB size

```
currDPBsize (for 4:2:0 format) =

(maxWidth * maxHeight)* 1.5*(1 +
(maxInterFrameInterval > 1));
```

❑ minBitrate need to be at least 10% lower than target bitrate ❑ minBitrate need to be at least 2 mbps lower than target bitrate For an example if 22 mbps is target average bitrate, minBitrate should be 19.8 mbps or lower. For an example if 10 mbps is target average bitrate, minBitrate should be 8 mbps or lower.

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 4-5

‖ 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. For interlaced case, it is height of one field.
inputWidth	XDAS_Int32	Input	Width of input frame in pixels
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.
targetBitRate	XDAS_Int32	Input	Target bit-rate in bits per second. For example, if the bit-rate is 2 Mbps, set this field to 2000000.
intraFrameInterval	XDAS_Int32	Input	Interval between two consecutive intra frames. For example: ❑ 0 - Only first frame to be intra coded ❑ 1 - No inter frames (all intra frames) ❑ N - One intra frame and N-1 inter frames, where N > 1.
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).

Field	Data Type	Input/ Output	Description
forceFrame	XDAS_Int32	Input	Force the current (immediate) frame to be encoded as a specific frame type. See enumeration IVIDEO_FrameType for more details
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). For example, this field will be: ❑ 0 - To use maxInterFrameInterval. ❑ 1 - Zero B frames between two reference frames. ❑ 2 - One B frame between two reference frames. ❑ 3 - Two B frames between two reference frames. and so on...
mvAccuracy	XDAS_Int32	Input	Pixel accuracy of the motion vector. See IVIDENC2_MotionVectorAccuracy enumeration in Table 4-1 for details.
sampleAspectRatioHeight	XDAS_Int32	Input	Sample aspect ratio height. This will be considered by encoder only when IH264ENC_VUICodingParams::aspectRatioIdc is IH264ENC_ASPECTRATIO_EXTENDED
sampleAspectRatioWidth	XDAS_Int32	Input	Sample aspect ratio width. This will be considered by encoder only when IH264ENC_VUICodingParams::aspectRatioIdc is IH264ENC_ASPECTRATIO_EXTENDED
ignoreOutbufSizeFlag	XDAS_Int32	Input	Flag to indicate that for bit-stream buffer size, application needs codec to expect the requested size or not Valid values are XDAS_TRUE and XDAS_FALSE.
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 pointer to receive buffer at sub-frame level
getBufferHandle	XDM_DataSyncHandle	Input	Handle that identifies the data sync FIFO and is passed as argument to getBufferFxn calls

Field	Data Type	Input/ Output	Description
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.

> Note:
>
> - The following are the limitations on the parameters of IVIDENC2_DynamicParams data structure:
>
>   inputHeight <= maxHeight
>
>   inputWidth <= maxWidth
>
> - See Table A.1 – Level Limits in ISO/IEC 14496-10 for the supported values of maxMbsPerSecond.
> - Use the following expression to calculate FrameSizeinMbs:
>
>   FrameSizeinMbs = (inputWidth * inputHeight) / 256;
>
> - Following condition should satisfy
>
>   maxMbsPerSecond >= FrameSizeinMbs*targetFrameRate

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

Field	Data Type	Input/ Output	Description
			(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 be achieved, See `IVIDENC2_Control and IH264ENC_Control` enumerations 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.

Field	Data Type	Input/ Output	Description
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: ❑ 1 - No B-frames ❑ 2 - Insert one B-frame. Not supported in this version of H264 Encoder ❑ N - Insert N-1 B frames between two P frames
inputChromaFormat	XDAS_Int32	Output	Chroma format for the input buffer. See XDM_ChromaFormat enumeration in Table 4-1 for details.
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 4-1 for details
profile	XDAS_Int32	Output	Profile indicator of video encoder. See IH264ENC_Profile enumeration for details
Level	XDAS_Int32	Output	Level indicator of video encoder. See IH264ENC_Level enumeration in Table 4-2 for details.
inputDataMode	XDAS_Int32	Output	Input data mode. See IVIDEO_DataMode enumeration n Table 4-1 for details.

Field	Data Type	Input/ Output	Description
outputDataMode	XDAS_Int32	Output	Output data Mode. See IVIDEO_DataMode enumeration n Table 4-1 for details.
numInputDataUnits	XDAS_Int32	Output	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.
numOutputDataUnit s	XDAS_Int32	Output	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.
configurationID	XDAS_Int32	Output	This is based on the codec configuration and can be used by the framework to optimize the save/restore overhead of any resources used.
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

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.
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 4-1 for details.
inputFrameSkip	XDAS_Int32	Output	Frame skipping modes for video. See IVIDEO_SkipMode enumeration in Table 4-1 for 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 given by application to encoder (through process call in IVIDEO2_BufDesc # planeDesc) 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 IVIDENC2_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 reconstruction buffer descriptor. See IVIDEO2_BufDesc data structure for more information These output buffers correspond to ❑ outBufs->bufs[1] ❑ outBufs->bufs[2] ❑ outBufs->bufs[3] reconBufs.bufDesc[0].buf is equivalent to outBufs->bufs[1] reconBufs.bufDesc[1].buf is equivalent to outBufs->bufs[2] reconBufs.bufDesc[2].buf is equivalent to outBufs->bufs[3]

Field	Data Type	Input/ Output	Description
			It is optional for encoder to populate this buffer descriptor. This implementation does not populate this descriptor.

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.

|| Fields

Field	Data Type	Input/ Output	Description
size	XDAS_Int32	Input/Output	Size of this structure
scatteredBlocksFlag	XDAS_Int32	Input/Output	Flag indicating whether the individual data blocks may be scattered in memory.
*baseAddr	XDAS_Int32	Input/Output	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,

Field	Data Type	Input/ Output	Description
			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 data blocks.
numBlocks	XDAS_Int32	Input/Output	Number of blocks available
varBlockSizesFlag	XDAS_Int32	Input/Output	Flag indicating whether any of the data blocks vary in size. Valid values are XDAS_TRUE and XDAS_FALSE.
*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 contains the size of same-size blocks. Memory for this array (of size numBlocks) has to be allocated by the caller of the putData API.

4.2.2 H.264 Encoder Data Structures

This section includes the following H.264 Encoder specific extended data structures:

- IH264ENC_Params
- IH264ENC_RateControlParams
- IH264ENC_InterCodingParams
- IH264ENC_IntraCodingParams
- IH264ENC_NALUControlParams
- IH264ENC_SliceCodingParams
- IH264ENC_LoopFilterParams
- IH264ENC_FMOCodingParams
- IH264ENC_DynamicParams
- IH264ENC_Inargs
- IH264ENC_Status
- IH264ENC_OutArgs
- IH264ENC_ProcessParams
- IH264ENC_ProcessParamsList
- IH264ENC_MetaDataFormatNaluInfo
- IH264ENC_MetaDataFormatUserDefinedSEI
- IH264ENC_Fxns
- IH264ENC_VUICodingParams
- IH264ENC_StereoInfoParms
- IH264ENC_FramePackingSEIParams
- IH264ENC_SVCCodingParams
- IH264ENC_ROIInput

4.2.2.1 IH264ENC_Params

|| Description

This structure defines the creation parameters and any other implementation specific parameters for a H.264 Encoder instance object. The creation parameters are defined in the XDM data structure, `IVIDENC2_Params`. For the default and supported values Table 4-13.

|| Fields

Field	Data Type	Input/ Output	Description
videnc2Params	IVIDENC2_Params	Input	See IVIDENC2_Params data structure for details.
rateControlParams	IH264ENC_RateControlParams	Input	Controls all rate control related parameters. See IH264ENC_RateControlParams data structure for details.
interCodingParams	IH264ENC_InterCodingParams	Input	Controls all inter coding related parameters. See IH264ENC_InterCodingParams data structure for details.
intraCodingParams	IH264ENC_IntraCodingParams	Input	Controls all intra coding related parameters. See IH264ENC_IntraCodingParams data structure for details.
nalUnitControlParams	IH264ENC_NALUControlParams	Input	Controls the insertion of different NALUs at different access points in video sequence. See IH264ENC_NALUControlParams data structure for details.
sliceCodingParams	IH264ENC_SliceCodingParams	Input	Controls all slice coding related parameters. See IH264ENC_SliceCodingParams data structure for details.
loopFilterParams	IH264ENC_LoopFilterParams	Input	Controls the in-loop filtering process. See IH264ENC_LoopFilterParams data structure for details.
fmoCodingParams	IH264ENC_FMOCodingParams	Input	Controls the FMO behavior. See IH264ENC_FMOCodingParams data structure for details.
vuiCodingParams	IH264ENC_VUICodingParams	Input	Controls the VUI parameters coding. See IH264ENC_VUICodingParams data structure for details.
stereoInfoParams	IH264ENC_StereoInfoParams	Input	Controls the stereo video coding. See IH264ENCStereoInfoParams data structure for details.
framePackingSEIParams	IH264ENC_FramePackingSEIParams	Input	Controls the frame packing SEI parameters for Stereo Video. See IH264ENC_FramePackingSEIParams data structure for details.

Field	Data Type	Input/ Output	Description
svcCodingParams	IH264ENC_SVCCodingParams	Input	Controls the SVC coding parameters. Refer Annex G of the H.264 standard for more details of SVC and parameters.
interlaceCodingType	XDAS_Int8	Input	Controls the type of interlaced coding. See IH264ENC_InterlaceCodingType enumeration in Table 4-2 for more details. If stereoInfoPreset != IH264_STEREOINFO_DISABLE && viewSelfContainedFlag == 0 then it gets overridden as IH264_INTERLACE_FIELDONLY_ARF If stereoInfoPreset != IH264_STEREOINFO_DISABLE && viewSelfContainedFlag == 1 then it gets overridden as IH264_INTERLACE_FIELDONLY_SPF
bottomFieldIntra	XDAS_Int8	Input	Controls the type of coding for second field for interlaced content
gopStructure	XDAS_Int8	Input	Defines the type of GOP structure, uniform and non-uniform. See IH264ENC_GOPStructure enumeration in Table 4-2 for more details.
entropyCodingMode	XDAS_Int8	Input	Controls the entropy coding type. See IH264ENC_EntropyCodingMode enumeration in Table 4-2 for more details.
transformBlockSize	XDAS_Int8	Input	Transform block size. See IH264ENC_TransformBlockSize enumeration in Table 4-2 for more details.
log2MaxFNumMinus4	XDAS_Int8	Input	Limits the maximum frame number in the bit-stream to (1<< (log2MaxFNumMinus4 + 4)) Range is 0 to12
picOrderCountType	XDAS_Int8	Input	Picture order count type. See IH264ENC_PicOrderCountType enumeration in Table 4-2 for more details.
enableWatermark	XDAS_Int8	Input	Enables or Disables water mark SEI message in the bit stream

Field	Data Type	Input/ Output	Description
IDRFrameInterval	XDAS_Int32	Input	Interval between two IDR frames, unit of this parameter is intraFrameInterval Example: ❑ 0 : Only first I frame as IDR ❑ 1 : All I frames are IDR. ❑ 2 : 1 out of 2 I frames are IDR starting from first I frame ❑ N: 1 out of N I frames are IDR starting from first frame When (numTemporalLayer > 1) then IDR frame will reset the temporal Gop structure and will start a new temporal Gop structure.
pConstantMemory	XDAS_Int32	Input	This pointer points to the memory area where constants are located. It has to be in DDR addressable space by vDMA. This is useful to allow re-locatable constants for the applications, which does not use Media Controller as host. Actual memory controller/allocator is on another master processor. If this is set to NULL then encoder assumes that all constants are pointed by symbol H264ENC_TI_ConstData
maxIntraFrameInterval	XDAS_Int32	Input	Maximum interval between two consecutive intra frames. For example: ❑ 0 - Only first frame to be intra coded ❑ 1 - No inter frames (all intra frames) ❑ N - One intra frame and N-1 inter frames, where N > 1.
debugTraceLevel	XDAS_UInt32	Input	This parameter configures the codec to dump a debug trace log ❑ 0 – No Trace is enabled ❑ 1 – Trace Level 1 is enabled ❑ 2 – Trace Level 2 is enabled ❑ 3 – Trace Level 3 is enabled
lastNFramesToLog	XDAS_UInt32	Input	This parameter configures the codec to maintain a history of last N frames/pictures. ❑ 0 – means only current frame trace is enabled ❑ 1 – means 1 previous frame trace is enabled apart from current frame ❑ N - means N previous frame trace is enabled apart from current frame
enableAnalyticinfo	XDAS_Int8	Input	This parameter configures the codec to expose analytic info like MVs and SAD parameters ❑ 0 – Disable ❑ Non-Zero - Enable

Field	Data Type	Input/ Output	Description
enableGMVSei	XDAS_Int8	Input	Enable or disable the TI specific GMV SEI message in the bit stream ❑ 0 – Disable ❑ Non-Zero - Enable
constraintSetFlags	XDAS_Int8	Input	This parameter controls the values of the constraint set flags in the bit stream. The flags that needs to be controlled are exposed as 4 lower bits of this byte. The 5th bit is the preset value that tells whether to use the default values of these flags as set by encoder or user defined values. The syntax of these bits is as below (MSB first) RESVD│ RESVD │ RESVD │ PRESET │CSF0│CSF1│CSF2 │CSF3 If the PRESET is set to zero then the values in the CSFX fields are ignored. If PRESET is 1 then encoder takes the values for CSF fields and codes in the bit stream.
enableRCDO	XDAS_Int8	Input	This parameter is used to enable encoding a bit stream compliant to Reduced Complexity Decoding Operations (RCDO) profile ❑ 0 – Disable ❑ Non-Zero - Enable
enableLongTermRefFrame	XDAS_Int32	Input	This parameter is used to support long-term reference frame. Setting this parameter equal to 1 will instruct encoder to keep its recent I/IDR frame in its reference buffer list. So it increases the DDR foot print by one frame buffer.
LTRPPeriod	XDAS_Int32	Input	This parameter is used to specify the long-term reference frame marking interval. This parameter is in use when enableLongTermRefFrame = IH264ENC_LTRP_REFERTOP_REACTIVE or IH264ENC_LTRP_REFERTO_PERIODICLTRP.
numTemporalLayer	XDAS_Int8	Input	This parameter controls the temporal Levels in bit-stream. 1 - Only base layer available in bit-stream. 2 - Maximum temporal level 1 in bit-stream 3 - Maximum temporal level 2 in bit-stream 4 - Maximum temporal level 3 in bit-stream
referencePicMarking	XDAS_Int8	Input	This parameter used to control the reference picture marking for any non-zero value means Long-term Picture (MMCO Commands) 0 - Short-term Picture (Sliding Window) 1 - Long-term Picture (MMCO Commands)

Field	Data Type	Input/ Output	Description
reservedParams[3]	XDAS_Int32	Input	Some part is kept reserved to add parameters later without changing the foot print of interface memory

Note:

Any field from the IH264ENC_Params (excluding IVIDENC2_Params) structure is useful only when the encodingPreset field of IVIDENC2_Params data structure is equal to XDM_USER_DEFINED.

constraintSetFlags: Care must be taken in setting the user defined values for the constrained set flags. The recommended settings are:

- Only in the base line profile the value of the CSF3 can be set to 1, If you want to convey the level as 1b. In all other cases it must be set to 0.
- In base line profile the values of CSF0, CSF1, CSF2 can be set to any values by application.
- In Main profile, the value of the CSF2 must be set to zero if you want to enable CABAC. It is recommended that this value is set to zero.
- In High Profile all the value of CSF should be zero as per standard.

4.2.2.2 IH264ENC_RateControlParams

|| Description

This structure controls rate control behavior. For the default and supported values, see Table 4-7.

|| Fields

Field	Data Type	Input/ Output	Description
rateControlParamsPreset	XDAS_Int8	Input	This preset controls the USER_DEFINED versus DEFAULT mode. If you are not aware about the fields, it should be set as IH264_RATECONTROLPARAMS_DEFAULT
scalingMatrixPreset	XDAS_Int8	Input	The preset controls between default, noisy, normal and std_default mode. It also allows for user to provide user defined scaling matrices at SPS/PPS level. If you are not aware about the fields, it should be set as IH264_SCALINGMATRIX_DEFAULT
rcAlgo	XDAS_Int8	Input	This defines the rate control algorithm to be used. Only useful if IVIDENC2::rateControlPreset is set as IVIDEO_USER_DEFINED

Field	Data Type	Input/ Output	Description
qpI	XDAS_Int8	Input	Initial quantization parameter for I/IDR frames. Valid Range is -1 to 51 -1 indicates auto initialization else Initial QP. When rateControlPreset = IVIDEO_NONE, this quantization parameter is used by the whole video frame/field.
qpMaxI	XDAS_Int8	Input	Maximum quantization parameter for I/IDR frame(s). Range is 0 to 51
qpMinI	XDAS_Int8	Input	Minimum quantization parameter for I/IDR frame(s). Range is 0 to 51.
qpP	XDAS_Int8	Input	Initial quantization parameter for P frames. Valid Range is -1 to 51 -1 indicates auto initialization else Initial QP. When rateControlPreset = IVIDEO_NONE, this quantization parameter is used by the whole video frame/field else qpP is decided by encoder internally. When rate control is enabled this parameter is used to encode the initial QP in PPS
qpMaxP	XDAS_Int8	Input	Maximum quantization parameter for inter frame(s). Range is 0 to 51.
qpMinP	XDAS_Int8	Input	Minimum quantization parameter for inter frame(s). Range is 0 to 51.
qpOffsetB	XDAS_Int8	Input	Offset of B frames Quantization Parameter from P frames and offset of the layer 1 frame's quantization parameter from the base layer in case of Hierarchical coding. qpP + qpOffsetB should be in range of [0,51]
qpMaxB	XDAS_Int8	Input	Maximum quantization parameter for B frame(s). Range is 0 to 51.
qpMinB	XDAS_Int8	Input	Minimum quantization parameter for B frame(s). Range is 0 to 51.
allowFrameSkip	XDAS_Int8	Input	Controls frame skip. ❑ 0 - Frame can never be skipped ❑ Non-zero - Frames can be skipped to achieve target bit-rate
removeExpensiveCoeff	XDAS_Int8	Input	Flag to remove high frequency expensive co-efficients.

Field	Data Type	Input/ Output	Description
chromaQPIndexOffset	XDAS_Int8	Input	Specifies offset to be added to luma Qp for addressing QpC values table for chroma components. Valid value is between -12 and 12, (inclusive)
IPQualityFactor	XDAS_Int8	Input	This provides configurality to control I frame quality with respect to P frame. Higher quality factor means I frame quality is given higher importance compared to P frame. See IH264ENC_FrameQualityFactor data structure for possible values.
initialBufferLevel	XDAS_Int32	Input	Initial buffer level for HRD compliance. It informs that hypothetical decoder can start depending on the fullness of the HRD buffer. Initial buffer level should be provided as absolute value of the buffer size.
HRDBufferSize	XDAS_Int32	Input	Hypothetical reference decoder 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 4096
minPicSizeRatioI	XDAS_Int16	Input	To determine ratio for min picture size. Allowed values are 1 to 4. Setting this to 0 will enable encoder to chosen theratio. User has to specify this value in Q5 format. minimum picture size is computed in the following manner, If minPicSizeRatioI is from 1 to 4 minPicSize = averagePicSize >> minPicSizeRatioI If minPicSizeRatioI is from 5 to 31 minPicSize = averagePicSize * minPicSizeRatioI >> Q5 Note that this is guided value to rate control to determine min picture size and encoder may not strictly follow this.
maxPicSizeRatioI	XDAS_Int16	Input	To determine ratio for max picture size. Allowed values are 2 to 30 & 33 to 960. Setting this to 0 and 1 will enable encoder to chosen the ratio. User has to specify this value in Q5 format. maximum picture size is computed in the following manner If maxPicSizeRatioI is from 2 to 30 maxPicSize = averagePicSize * maxPicSizeRatioI If maxPicSizeRatioI is from 33 to 960 maxPicSize = averagePicSize *

Field	Data Type	Input/ Output	Description
			maxPicSizeRatioI >> Q5 Note that this is guided value to rate control to determine max picture size and encoder may not strictly follow this.
minPicSizeRatioP	XDAS_Int16	Input	To determine ratio for min picture size for P pictures. Details similar to minPicSizeRatioI
maxPicSizeRatioP	XDAS_Int16	Input	To determine ratio for max picture size for P pictures. Details similar to maxPicSizeRatioI
minPicSizeRatioB	XDAS_Int16	Input	To determine ratio for min picture size for B pictures. Details similar to minPicSizeRatioI
maxPicSizeRatioB	XDAS_Int16	Input	To determine ratio for max picture size for B pictures. Details similar to maxPicSizeRatioI
enablePRC	XDAS_Int8	Input	Control flag to enable MB level perceptual rate control
enablePartialFrameSkip	XDAS_Int8	Input	Control flag to enable partial frame skip. Only useful with CBR rate control mode
discardSavedBits	XDAS_Int8	Input	Control Flag to discard saved bits for future pictures. In VBR ratecontrol mode, the saved bits in low complexity scenes will be used for future scene/pictures With this flag 0, encoder will use saved bits for future scenes and for any non-zero value encoder discards the saved bits. Only useful with VBR ratecontrol mode.
reserved	XDAS_Int8	Input	Some part is maintained as reserved to add parameters later without changing the foot print of interface memory
VBRDuration	XDAS_Int32	Input	Duration over which statistics are collected to switch bit-rate states.Increasing this value will make VBR wait for longer time before switching bit-rate state
VBRsensitivity	XDAS_Int8	Input	Specifies the target bitrate used by rate control in high complexity state.
skipDistributionWindowLength	XDAS_Int16	Input	Number of frames over which the skip frames can be distributed
numSkipInDist	XDAS_Int16	Input	Number of skips allowed within the distribution

Field	Data Type	Input/ Output	Description
ributionWindow			window.
enableHRDComplianceMode	XDAS_Int8	Input	It controls compliance of H.264 encoder to HRD model specified in H.264 annexure C trace level 0 : no strict hrd compliance, good for better quality Non-Zero : RC behavior complaint to standard defined HRD
frameSkipThMulQ5	XDAS_Int32	Input	Frame skip threshold in Q5 format. It is computed based on thefollowing equation frameSkipThreshold = HRDBufferSize – (frameSkipThMulQ5 * averagePicSize) >>5 0 : Encoder chosen value [16,128] : User defined value which will override encoder chosen value
vbvUseLevelThQ5	XDAS_Int32	Input	VBV use level in Q5 format. Vbv buffer use level is computed based on the following equation vbvUseLevel = (vbvUseLevelThQ5 * averagePicSize)>>5 0 : Encoder chosen value [16,128] : User defined value which will override encoder chosen value
ReservedRC[3]	XDAS_Int32	Input	Some part is maintained as reserved to add parameters later without changing the foot print of interface memory

Note:

- With enablePartialFrameSkip = non-zero, encoder might not respect the qpMax constraints. Encoded bit-streams might have macro blocks with QP > qpMax for any picture type.
- In VBR rate control algorithm, with a scene change the frame having scene change will follw qpMaxI and qpMinI irrespective of frame type

4.2.2.3 IH264ENC_InterCodingParams

|| Description

This structure contains all the parameters which controls inter MBs coding behavior. For the default and supported values, see. Table 4-8

|| Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
interCodingPreset	XDAS_Int8	Input	This preset controls the USER_DEFINED versus DEFAULT mode. If you are not aware about the fields, it should be set as IH264_INTERCODING_DEFAULT
searchRangeHorP	XDAS_Int16	Input	Horizontal search range for P frames. Possible values: Non zero, maximum up to 144
searchRangeVerP	XDAS_Int16	Input	Vertical search range for P frames. Possible Values: Non-zero, maximum up to 32
searchRangeHorB	XDAS_Int16	Input	Horizontal search range for B frames. Possible values: Non zero, maximum up to 144
searchRangeVerB	XDAS_Int16	Input	Vertical search range for B frames. Possible values: Non-zero, maximum up to 16
interCodingBias	XDAS_Int8	Input	Bias control for having a macro block coded as inter or intra See IH264ENC_BiasFactor enumeration in Table 4-2 for possible values
skipMVCodingBias	XDAS_Int8	Input	Bias control for having a macro block use skip MV or regular MV. See IH264ENC_BiasFactor enumeration in Table 4-2 for possible values
minBlockSizeP	XDAS_Int8	Input	Minimum block size for P frames. See IH264ENC_InterBlockSize enumeration in Table 4-2 for possible values
minBlockSizeB	XDAS_Int8	Input	Minimum block size for B frames. See IH264ENC_InterBlockSize enumeration in Table 4-2 for possible values
meAlgoMode	XDAS_Int8	Input	Motion Estimation algorithm Mode. See IH264ENC_MeAlgoMode in Table 4-2 for possible values.

Note:

- None of the parameters is ignored during run-time.
- meAlgoMode = IH264ENC_MOTIONESTMODE_HIGH_SPEED is supported only with interFrameInterval = 1, minBlockSizeP = IH264_BLOCKSIZE_16x16 and mvAccuracy = IVIDENC2_MOTIONVECTOR_QUARTERPEL
- When meAlgoMode is slected as IH264ENC_MOTIONESTMODE_HIGH_SPEED along with transformBlockSize == IH264_TRANSFORM_8x8 in P picture then a performance customized flow is enabled in encoder for mode decision. In that flow IH264ENC_IntraCodingParams :: intraRefreshMethod parameter is ignored. In the same customized flow IH264ENC_RateControlParams :: enablePartialFrameSkip is also ignored.

4.2.2.4 IH264ENC_IntraCodingParams

|| Description

This structure defines all the operations on H.264 Encoder instance objects. For the default and supported values, see Table 4-9.

|| Fields

Field	Data Type	Input/ Output	Description
intraCodingPreset	XDAS_Int8	Input	This preset controls the user defined versus default mode. If you are not aware about the fields, it should be set as INTRA_CODING_DEFAULT, other wise INTRA_CODING_USER_DEFINED.
lumaIntra4x4Enable	XDAS_Int16	Input	This parameter controls the Luma Intra4x4 encoding in video encoder. A bit-field is provided for each Luma intra4x4 mode as shown: HOR_UP|VERT_LEFT|HOR_DOWN|VERT_RIGHT|DIAG_DOWN_RIGHT|DIAG_DOWN_LEFT|DC|HOR|VER Set/ reset particular bit to enable/disable that mode (0=disable, 1=enable) DC (bit-2) is ignored Bit-10 and above are ignored
lumaIntra8x8Enable	XDAS_Int16	Input	This parameter controls the Luma Intra8x8 encoding in video encoder. A bit-field is given for each Luma intra8x8 mode as shown: HOR_UP|VERT_LEFT|HOR_DOWN|VERT_RIGHT|DIAG_DOWN_RIGHT|DIAG_DOWN_LEFT|DC|HOR|VER Set/ reset particular bit to enable/disable that mode (0=disable, 1=enable) DC (bit-2)is ignored For example : 139(decimal) = 0x8B = 010001011 (bits) = HOR, VER, VERT_LEFT are enabled and DC is always enabled. Bit-10 and above are ignored
lumaIntra16x16Enable	XDAS_Int8	Input	This parameter controls the Luma Intra16x16 encoding in video encoder. A bit-field is given for each Luma intra16x16 mode as shown: PLANE|DC|HOR|VER Set/ reset particular bit to enable/disable that mode (0=disable, 1=enable) DC (bit-2)is ignored Bit-4 and above are ignored
chromaIntra8x8Enable	XDAS_Int8	Input	This parameter controls the chroma Intra8x8 encoding in video encoder. A bit-field is given for each chroma intra8x8 mode as shown:

Field	Data Type	Input/ Output	Description
			PLANE│VER│HOR│DC Set/ reset particular bit to enable/disable that mode (0=disable, 1=enable) DC (bit-0) is ignored Bit-4 and above are ignored
chromaComponentEnable	XDAS_Int8	Input	This parameter controls the chroma intra prediction search. You can choose to perform chroma intra estimation for both Cb and Cr samples or only on Cr samples. For more details, see IH264ENC_ChormaComponent enumeration in Table 4-2.
intraRefreshMethod	XDAS_Int8	Input	Mechanism to do intra refresh. See IH264ENC_IntraRefreshMethods enumeration in Table 4-2 for possible values
intraRefreshRate	XDAS_Int16	Input	Rate at which intra refresh is done. Case : intraRefreshMethod = IH264_INTRAREFRESH_CYCLIC_MBS This rate is specified as One IntraMB per # MBs. For example if rate is 20, there has to be one intra MB(s) per 20 Mbs. Case : intraRefreshMethod = IH264_INTRAREFRESH_GDR Intra Refesh Rate is treated as the number of rows to be intra refreshed per frame.
gdrOverlapRowsBtwFrames	XDAS_Int16	Input	Defines the Overlap of the Intra Refresh Region between successive frame in case the intraRefreshMethod IH264_INTRAREFRESH_GDR or else treated to be don't care. Again gdrOverlapRowsBtwFrames should be less than intraRefreshRate
constrainedIntraPredEnable	XDAS_Int16	Input	Controls the intra macroblock coding in P slices. Valid values are 0,non-zero
intraCodingBias	XDAS_Int8	Input	Controls percentage of intra macroblocks. Refer IH264ENC_IntraCodingBias for supported values. This control is usefull to tune the HDVICP 2.0 utilization.

Note:

- transformBlockSize is applicable only for inter MBs
- If transformBlockSize == IH264_TRANSFORM_8x8 then encoder will only use 8x8 transform for INTER coded MBs
- If transformBlockSize == IH264_TRANSFORM_4x4 then encoder

will only use 4x4 transform for INTER coded MBs ❑ If `transformBlockSize` == `IH264_TRANSFORM_ADAPTIVE` then encoder will decide transform size adaptively at MB-level. ❑ Intra refresh mechanism IH264_INTRAREFRESH_GDR is not supported for interlaced sequences and for 'B' frame cases.

4.2.2.5 IH264ENC_NALUControlParams

|| Description

This structure contains all the parameters that define the control mechanism for insertion of different NALU types at different point in video sequence. For the default and supported values, see Table 4-10.

|| Fields

Field	Data Type	Input/ Output	Description
`naluControlPreset`	`XDAS_Int16`	Input	This preset controls the user defined versus default mode. If you are not aware about the fields, it should be set as `IH264_NALU_CONTROL_DEFAULT` other wise `IH264_NALU_CONTROL_USERDEFINED`

`naluPresentMaskStartOfSequence`	`XDAS_Int16`	Input	This parameter controls the insertion of different NALU at start of sequence
			A bit-field is given for each NALU type as shown.
			
			`14| 13| 12| 11| 10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0`
			`UD_SEI|SPS+VUI|FILLER|EOSTREAM|EOSEQ|AUD|PPS|SPS|SEI|IDR_SLICE|SLICE_DP_C|SLICE_DP_B|SLICE_DP_A|SLICE|UNSPECIFIED`
			
			Set/reset particular bit to enable/disable that insertion of that NALU (0=disable, 1=enable)
			`SLICE_DP_A`(bit-2), `SLICE_DP_B`(bit-3), `SLICE_DP_C`(bit-4), `SPS_EXT`(bit-13) is ignored and assumed to be zero.
			EOSEQ(bit-10), EOSTREAM(bit-11) is ignored and assumed to be zero.
			bits 0-5 are ignored See Appendix B for details.

`naluPresentMaskIDRPicture`	`XDAS_Int16`	Input	This parameter controls the insertion of different NALU at IDR picture
			A bit-field is given for each NALU type as shown:
			
			`14| 13| 12| 11| 10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0`
			`UD_SEI|SPS+VUI|FILLER|EOSTREAM|EOSEQ|AUD|PPS|SPS|SEI|IDR_SLICE|SLICE_DP_C|SLICE_DP_B|SLICE_DP_A|SLICE|UNSPECIFIED`
			
			Set/ reset particular bit to enable/disable that insertion of that NALU (0=disable, 1=enable)
			`SLICE_DP_A`(bit-2), `SLICE_DP_B`(bit-3), `SLICE_DP_C`(bit-4), `SPS_EXT`(bit-13) is ignored and assumed to be zero

Field	Data Type	Input/ Output	Description
			EOSEQ(bit-10), EOSTREAM(bit-11) is ignored and assumed to be zero bits 0-5 are ignored See Appendix B for details.

naluPresentMaskIntraPicture	XDAS_Int16	Input	This parameter controls the insertion of different NALU at Intra picture(s). A bit-field is given for each NALU type as shown:
			
			`14| 13| 12| 11| 10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0`
			`UD_SEI|SPS+VUI|FILLER|EOSTREAM|EOSEQ|AUD|PPS|SPS|SEI|IDR_SLICE|SLICE_DP_C|SLICE_DP_B|SLICE_DP_A|SLICE| UNSPECIFIED`
			
			Set/ reset particular bit to enable/disable that insertion of that NALU (0=disable, 1=enable)
			`SLICE_DP_A`(bit-2), `SLICE_DP_B`(bit-3), `SLICE_DP_C`(bit-4), `SPS_EXT`(bit-13) is ignored and assumed to be zero
			EOSEQ(bit-10), EOSTREAM(bit-11) is ignored and assumed to be zero
			bits 0-5 are ignored See Appendix B for details.

naluPresentMaskNonIntraPicture	XDAS_Int16	Input	This parameter controls the insertion of different NALU at Non-intra pictures
			A bit-field is given for each NALU type as shown:
			
			`14| 13| 12| 11| 10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0`
			`UD_SEI|SPS+VUI|FILLER|EOSTREAM|EOSEQ|AUD|PPS|SPS|SEI|IDR_SLICE|SLICE_DP_C|SLICE_DP_B|SLICE_DP_A|SLICE| UNSPECIFIED`
			
			Set/ reset particular bit to enable/disable that insertion of that NALU (0=disable, 1=enable)
			`SLICE_DP_A`(bit-2), `SLICE_DP_B`(bit-3), `SLICE_DP_C`(bit-4), `SPS_EXT`(bit-13) is ignored and assumed to be zero
			EOSEQ(bit-10), EOSTREAM(bit-11) is ignored and assumed to be zero.
			bits 0-5 are ignored See Appendix B for details.

naluPresentMaskEndOfSequence	XDAS_Int16	Input	This parameter controls the insertion of different NALU at end of sequence
			A bit-field is given for each NALU type as shown:
			
			`14| 13| 12| 11| 10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0`
			`UD_SEI|SPS+VUI|FILLER|EOSTREAM|EOSEQ|AUD|PPS|SPS|SEI|IDR_SLICE|SLICE_DP_C|SLICE_DP_B|SLICE_DP_A|SLICE| UNSPECIFIED`
			
			Set/ reset particular bit to enable/disable that insertion of that NALU (0=disable, 1=enable)
			Except bit-11 and bit-12, rest all bits are ignored

Field	Data Type	Input/ Output	Description
			and assumed to be zero. See Appendix B for details See Appendix B for details.

4.2.2.6 IH264ENC_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`	`XDAS_Int8`	Input	This preset controls the user defined versus default mode. If you are not aware about the fields, it should be set as `IH264_SLICECODING_DEFAULT`
`sliceMode`	`XDAS_Int16`	Input	This defines the control mechanism to split a picture in slices. It can be a> Single slice per picture. b> MB based. c> Bytes based(H241) or d> Offset based in units of rows. Restriction for H241: The `sliceMode` 2 is supported only if the frame width is more than 128 and `sliceUnitSize` value should be greater than 256.
`sliceUnitSize`	`XDAS_Int16`	Input	❑ If `sliceMode` == `IH264_SLICEMODE_MBUNIT`, then this parameter informs the number of macro blocks in one slice ❑ If `sliceMode` == `IH264_SLICEMODE_BYTES`, then this parameter informs the number of bytes in one slice ❑ If `sliceMode` == `IH264_SLICEMODE_OFFSET`, then this parameter informs the number of offset information provided by user. Actual offset are provided with `sliceRowStartNumber` parameter.
`sliceStartOffset[IH264ENC_MAX_NUM_SLICE_START_OFFSET]`	`XDAS_Int8`	Input	Row numbering is assumed to start from 0. Entries in this array must have numbers in ascending order. First slice of the picture is always starting from 0th row of the picture, so 0th entry is the offset of second slice in picture. ❑ Example 1: `sliceStartOffset[0]` = 25 , `sliceStartOffset [1]` = 30, `sliceStartOffset [2]` = 40 will result into 4 slices starting from row# 0, 25, 30 and 40

Field	Data Type	Input/ Output	Description
			❑ Example 2: `sliceStartOffset [0]` = 25 , `sliceStartOffset [1]` = 70, `sliceStartOffset [2]` = 60 is invalid ❑ Example 3: `sliceStartOffset [0]` = 25 , `sliceStartOffset [1]` = 50, `sliceStartOffset [2]` = 100 will result into 3 slices starting from row# 0, 25 and 50 (if number of rows in picture < (100 + 1))
`streamFormat`	`XDAS_Int8`	Input	Controls the type of stream: byte stream format or NALU format See `IH264ENC_StreamFormat` enumeration in enumeration in Table 4-2 for possible values

4.2.2.7 IH264ENC_LoopFilterParams

‖ Description

This structure contains all the parameters, which controls loop filtering operations. For the default and supported values, see Table 4-12.

‖ Fields

Field	Data Type	Input/ Output	Description
`loopfilterPreset`	`XDAS_Int8`	Input	This preset controls the user defined versus default mode. If you are not aware about the fields, it should be set as `IH264_SLICECODING_DEFAULT`
`loopfilterDisableIDC`	`XDAS_Int8`	Input	Controls H.264 loop filter disabling options
`filterOffsetA`	`XDAS_Int8`	Input	Alpha offset for loop filter Range is [-12, 12] even number
`filterOffsetB`	`XDAS_Int8`	Input	Beta offset for loop filter Range is [-12, 12] even number

4.2.2.8 IH264ENC_FMOCodingParams

‖ Description

This structure contains all the parameters which controls FMO operations. For the default and supported values, see Table 4-13.

‖ Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
fmoCodingPreset	XDAS_Int8	Input	This preset controls the user defined versus default mode. If you are not aware about the fields, it should be set as IH264_SLICECODING_DEFAULT
numSliceGroups	XDAS_Int8	Input	Total number of slice groups. Valid values are [0,8]
sliceGroupMapType	XDAS_Int8	Input	Type of slice group. See IH264ENC_SliceGroupMapType enumeration in Table 4-2 for possible values.
sliceGroupChangeDirectionFlag	XDAS_Int8	Input	Only valid when sliceGroupMapType is equal to IH264_RASTER_SCAN_SLICE_GRP, IH264_WIPE_SLICE_GRP or IH264_WIPE_SLICE_GRP. See IH264ENC_SliceGroupChangeDirection enumeration in Table 4-2 for possible values
sliceGroupChangeRate	XDAS_Int8	Input	Only valid when sliceGroupMapType is equal to IH264_RASTER_SCAN_SLICE_GRP, IH264_WIPE_SLICE_GRP or IH264_WIPE_SLICE_GRP Valid values are : [0, factor of number of Mbs in a row]
sliceGroupChangeCycle	XDAS_Int16	Input	Only valid when sliceGroupMapType is equal to IH264_RASTER_SCAN_SLICE_GRP, IH264_WIPE_SLICE_GRP or IH264_WIPE_SLICE_GRP Valid values can be 0 to numMbsRowsInPicture, also constrained by sliceGroupChangeRate*sliceGroupChangeCycle < totalMbsInFrame
sliceGroupParams[MAXNUMSLCGPS]	XDAS_Int16	Input	This field is useful when sliceGroupMapType is equal to either IH264_INTERLEAVED_SLICE_GRP or IH264_FOREGRND_WITH_LEFTOVER_SLICE_GRP In case of IH264_INTERLEAVED_SLICE_GRP, the i-th entry in this array is used to specify the number of consecutive slice group macro blocks to be assigned to the i-th slice group in raster scan order of slice group macro block units. Valid values are 0 to totalMbsInFrame again constrained by sum of all the elements should not exceed totalMbsInFrame In case of IH264_FOREGRND_WITH_LEFTOVER_SLICE_GRP: ❑ First entry in the array specify the start position of foreground region in terms of

Field	Data Type	Input/ Output	Description
			macro block number. Valid values are [0, totalMbsInFrame-1]. ❑ Second entry in the array specifies the end position of foreground region in terms of macro block number. Valid values are [0, totalMbsInFrame-1] with following constrains: `endPos > startPos && endPosmbsInOneRow > startPosmbsInOneRow`

4.2.2.9 IH264ENC_DynamicParams

|| Description

This structure defines the run-time parameters and any other implementation specific parameters for a H.264 Encoder instance object. The run-time parameters are defined in the XDM data structure, `IVIDENC2_DynamicParams`. For the default and supported values, see Table 4-19.

|| Fields

Field	Data Type	Input/ Output	Description
`videnc2DynamicParams`	`IVIDENC2_DynamicParams`	Input	See `IVIDENC2_DynamicParams` data structure for details.
`rateControlParams`	`IH264ENC_RateControlParams`	Input	Controls all rate control related parameters. Only few are supported to be changed as part control call. See `IH264ENC_RateControlParams` data structure for more details.
`interCodingParams`	`IH264ENC_InterCodingParams`	Input	Controls all inter MB coding related parameters. Only few are supported to be changed as part control call. See `IH264ENC_InterCodingParams` data structure for more details
`intraCodingParams`	`IH264ENC_IntraCodingParams`	Input	Controls all intra coding related parameters. Only few are supported to be changed as part control call. See `IH264ENC_IntraCodingParams` data structure for details.

Field	Data Type	Input/ Output	Description
sliceCodingParams	IH264ENC_SliceCodingParams	Input	Controls all slice coding related parameters. Only few are supported to be changed as part control call. See IH264ENC_SliceCodingParams data structure for more details.
sliceGroupChangeCycle	XDAS_Int32	Input	Only valid when sliceGroupMapType is equal to IH264_RASTER_SCAN_SLICE_GRP, IH264_WIPE_SLICE_GRP or IH264_WIPE_SLICE_GRP Valid values can be 0 to numMbsRowsInPicture, also constrained by sliceGroupChangeRate*sliceGroupChangeCycle < totalMbsInFrame Only valid when sliceGroupMapType is equal to IH264_RASTER_SCAN_SLICE_GRP. Valid values are : [0, factor of number of Mbs in a row]
searchCenter	XDM_Point	Input	Search center for motion estimation. XDM_Point.x == 0x7FFF means ignore searchCenter
enableStaticMBCount	XDAS_Int8	Input	Flag to indicate enable/disable of H.241 defined Static MB count ❑ 0 – Disable ❑ Non-Zero - Enable
enableROI	XDAS_Int32	Input	Flag to Enable/Disable ROI coding. ❑ Non-Zero – enable ROI coding. ❑ 0 – disable ROI coding. Default value = 0. ROI will be automatically disabled in case of full frame skip and for skip macroblocks.
reservedDynParams[3]	XDAS_Int32	Input	Some part is maintained as reserved to add parameters later without changing the foot print of interface memory

Note:

Any field from the IH264ENC_DynamicParams excluding IVIDENC2_DynamicParams) structure is useful only when the encodingPreset field of IVIDENC2_Params data structure is equal to XDM_USER_DEFINED.

4.2.2.10 IH264ENC_InArgs

|| Description

This structure defines the run-time input arguments for H.264 Encoder instance object.

|| Fields

Field	Data Type	Input/ Output	Description
videnc2InArgs	IVIDENC2_InArgs	Input	See IVIDENC2_InArgs data structure for details
processId	XDAS_Int32	Input	processId in InArgs was kept to ease the acquire time optimization in application code. In N channel case, acquire is happening for last channel and this (processId) as argument is passed into acquire call. This will make application to understand that for which process call, acquire has been made. With this information application can optimize the time spent in acquire. Like, it might have happened that from last call of acquire, HDVICP2 became unavailable to any further process call(s). In this scenario application will get to know that HDVICP2 was not given to somebody else from last process call, and hence it can do some optimization in acquire routine.
roiInputParams	IH264ENC_RoiInput	Input	This is to pass the ROI related data to the algorithm. See IH264ENC_RoiInput data structure (Section 4.2.2.11) for details.
inputKey	XDAS_UInt32	Input	This parameter along with the few important properties of a frame are used to generate the encrypted key. If watermarking is enabled then this encrypted key would be inserted in the form of user data unregistered SEI message in the encoded stream

4.2.2.11 IH264ENC_RoiInput

|| Description

This structure defines the run-time ROI related input information for H.264 Encoder instance object.

‖ **Fields**

Field	Data Type	Input/ Output	Description
`listROI [IH264ENC_MAX_ROI]`	`XDM_Rect`	Input	This gives the location of each ROI in terms of top left and bottom right (x,y) co-ordinates.
`roiType[IH264ENC_MAX_ROI]`	`XDAS_Int8`	Input	Type of each ROI. The supported types are FACE_OBJECT, BACKGROUND_OBJECT, FOREGROUND_OBJECT, DEFAULT_OBJECT,**and** PRIVACY_MASK .
`numOfROI`	`XDAS_Int8`	Input	Number of ROIs in the current frame.
`roiPriority[IH264ENC_MAX_ROI]`	`XDAS_Int32`	Input	Holds the priority information of each ROI. Valid values include all integers between -8 and 8, inclusive. A higher value means that more importance will be given to the ROI compared to other regions. In other words, it determines the number of bits given to ROI. This parameter holds the mask color information if ROI is of type privacy mask. In fixed Qp mode, This filed holds the Qp value of specified ROI.

> Note:
>
> This encoder supports a maximum of 36 ROIs in a frame i.e., `IH264ENC_MAX_ROI` is 36.
>
> Overlapping of ROIs of *different* ROI type is not allowed.
>
> If the ROI is detected as `FACE_OBJECT`, then a guard band is added around it. For all other ROI types, no guard band is added.

4.2.2.12 IH264ENC_Status

‖ **Description**

This structure defines parameters that describe the status of the H.264 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 h264 encoder along with error information, if any.
rateControlParams	IH264ENC_RateControlParams	Output	See IH264ENC_RateControlParams data structure for details.
interCodingParams	IH264ENC_InterCodingParams	Output	See IH264ENC_InterCodingParams data structure for details.
intraCodingParams	IH264ENC_IntraCodingParams	Output	See IH264ENC_IntraCodingParams data structure for details.
nalUnitControlParams	IH264ENC_NALUControlParams	Output	See IH264ENC_NALUControlParams data structure for details.
sliceCodingParams	IH264ENC_SliceCodingParams	Output	See IH264ENC_SliceCodingParams data structure for details.
loopFilterParams	IH264ENC_LoopFilterParams	Output	See IH264ENC_LoopFilterParams data structure for details.
fmoCodingParams	IH264ENC_FMOCodingParams	Output	See IH264ENC_FMOCodingParams data structure for details.
vuiCodingParams	IH264ENC_VUICodingParams	Output	See IH264ENC_VUICodingParams data structure for details.
stereoInfoParams	IH264ENC_StereoInfoParams	Output	See IH264ENC_StereoInfoParams structure for details.
framePackingSEIParams	IH264ENC_FramePackingSEIParams	Output	See IH264ENC_FramePackingSEIParams structure for details.
svcCodingParams	IH264ENC_SVCCodingParams	Output	See IH264ENC_SVCCodingParams structure for details.
interlaceCodingType	IH264ENC_InterlaceCodingType	Output	See IH264ENC_InterlaceCodingType enumeration in Table 4-2 for details.
bottomFieldIntra	XDAS_Int8	Output	Controls the type of coding for bottom field for interlaced content
gopStructure	IH264ENC_GOPStructure	Output	See IH264ENC_GOPStructure enumeration in Table 4-2 for details
entropyCodingMode	IH264ENC_EntropyCodingMode	Output	See IH264ENC_EntropyCodingMode enumeration in Table 4-2 for details.
transformBlockSize	IH264ENC_TransformBlockSize	Output	See IH264ENC_TransformBlockSize enumeration in Table 4-2 for details.

Field	Data Type	Input/ Output	Description
log2MaxFNumMinus4	XDAS_Int8	Output	Limits the maximum frame number in the bit-stream to (1<< (log2MaxFNumMinus4 + 4)) Range is 0 to 12.
picOrderCountType	IH264ENC_PicOrderCountType	Output	See IH264ENC_PicOrderCountType enumeration in Table 4-2 for details.
enableWatermark	XDAS_Int8	Output	This Parameter tells if WaterMark SEI messages is enabled or disabled in bitstream 0 – Disable, Non-Zero - Enable
IDRFrameInterval	XDAS_Int32	Output	Interval betweenw two IDR frames, it should be and integer multiple of intraFrameInterval
maxIntraFrameInterval	XDAS_Int32	Output	Maximum Interval between two consecutive intra frames. For example: ❑ 0 - Only first frame to be intra coded ❑ 1 - No inter frames (all intra frames) N - One intra frame and N-1 inter frames, where N > 1.
debugTraceLevel	XDAS_UInt32	Output	Level of trace
lastNFramesToLog	XDAS_UInt32	Output	Number of previous pictures for which trace is available
enableAnalyticinfo	XDAS_Int8	Output	This parameter configures the codec to expose analytic info like MVs and SAD parameters ❑ 0 – Disable ❑ Non-Zero – Enable
enableGMVSei	XDAS_Int32	Output	Enable or disable the TI specific GMV SEI message in the bit stream ❑ 0 – Disable ❑ Non-Zero - Enable
constraintSetFlags	XDAS_Int8	Output	This parameter controls the values of the constraint set flags in the bit stream. The flags that needs to be controlled are exposed as 4 lower bits of this byte. The 5th bit is the preset value that tells whether to use the default values of these flags as set by encoder or user defined values. The syntax of these bits is as below (MSB first) RESVD| RESVD | RESVD | PRESET |CSF0|CSF1|CSF2 |CSF3 If the PRESET is set to zero then the values in the CSFX fields are ignored. If PRESET is 1 then encoder takes the values for CSF fields and codes in the bit stream.

Field	Data Type	Input/ Output	Description
enableRCDO	XDAS_Int8	Output	This parameter is used to enable encoding a bit stream compliant to Reduced Complexity Decoding Operations (RCDO) profile ❑ 0 – Disable ❑ Non-Zero – Enable
enableLongTermRefFrame	XDAS_Int8	Output	This parameter is used to support long-term reference frame. Setting this parameter equal to 1 will instruct encoder to keep its recent I/IDR frame in its reference buffer list. So it increases the DDR foot print by one frame buffer.
LTRPPeriod	XDAS_Int32	Output	This parameter is used to specify the long-term reference frame marking interval. This parameter is in use when enableLongTermRefFrame = IH264ENC_LTRP_REFERTOP_REACTIVE or IH264ENC_LTRP_REFERTO_PERIODICLTRP.
searchCenter	XDM_Point	Output	See XDM_Point data structure for details.
enableStaticMBCount	XDAS_Int8	Output	Flag to indicate enable/disable of H.241 defined Static MB count ❑ 0 – Disable ❑ Non-Zero - Enable
extMemoryDebugTraceAddr	XDAS_UInt32	Output	Address in external memory where the trace data is available
numTemporalLayer	XDAS_Int8	Output	This parameter controls the temporal levels in bit-stream.
referencePicMarking	XDAS_Int8	Output	This parameter used to control the reference picture marking.
extMemoryDebugTraceSize	XDAS_UInt32	Output	Size of the trace data
enableROI	XDAS_Int8	Output	Flag to indicate enable/disable ROI coding. ❑ Non-Zero – enable ROI coding. ❑ 0 – disable ROI coding.

4.2.2.13 IH264ENC_OutArgs

|| Description

This structure defines the run-time output parameters for the H.264 Encoder instance object.

|| Fields

Field	Data Type	Input/ Output	Description
videnc2OutArgs	IVIDENC2_OutArgs	Output	See IVIDENC2_OutArgs data structure for details.
bytesGeneratedBotField	XDAS_Int32	Output	Number of bytes generated for bottom field during the IVIDENC2_Fxns::process() call. This field is updated only in case of contentType = Interlaced and both the fields are provided to codec in single process call
vbvBufferLevel	XDAS_Int32	Output	This variable tells the buffer level at the end of every picture coding from decoder perspective. The value populated in this variable is latest for every process call
numStaticMBs	XDAS_Int32	Output	Number of static MBs (defined by H241) in encoded picture during the IVIDENC2_Fxns::process() call. This field is valid only if dynamicParams.enableStaticMBCount is set to non-zero.
temporalId	XDAS_Int32	Output	This parameter carries the temporal layer Id of current frame in Hierarchical encoding. ❑ If the value of IH264ENC_Params::numTemporalLayer parameter is 1 (IH264_TEMPORAL_LAYERS_1, base layer encoding) then its value is 0 for P-pictures and 1 for B-pictures. ❑ If the value of IH264ENC_Params::numTemporalLayer parameter is more than 1 (H-P encoding) then this parameter holds the temporal layer id of the current picture. ❑ In case of interlace, both the fields will have the same temporal id. ❑ If the value of IH264ENC_SVCCodingParams::svcExtensionFlag is set (IH264_SVC_EXTENSION_FLAG_ENABLE), then the bit-stream will have the SSPS,prefix-NALU. The temporal_id value encoded in the prefix-NALU and the value of this parameter are same.
control	XDAS_Int32	Output	Encoder control operations. Most of the times it is IVIDENC2_InArgs::control. But there are certain cases when it is not same as IVIDENC2_InArgs::control, hence it is advisable to look at this output information.

Note:

Interpretation of bytesGenerated field depends upon usage of base/extended class.

If Base class of `OutArgs` only:

- `outArgs->bytesGenerated` will have bytes generated of a frame for progressive case
- `outArgs->bytesGenerated` will have sum of bytes generated for both field if single process call is made for both the fields (interlaced case)
- `outArgs->bytesGenerated` will have bytes generated for each field if single process call is made for each field (interlaced case)

If Extended class of OutArgs only:

- `outArgs->bytesGenerated` will have bytes generated of a frame for progressive case
- `outArgs->bytesGenerated` will have sum of bytes generated for both field if single process call is made for both the fields and `outargsextended->bytesGeneratedBottomField` will have bytes generated for bottom field (interlaced case)
- `outArgs->bytesGenerated` will have bytes generated for each field if single process call is made for each field (interlaced case)

4.2.2.14 IH264ENC_ProcessParams

|| Description

This structure defines the container for holding the channel information.

|| Fields

Field	Data Type	Input/ Output	Description
handle	IVIDENC2_Handle	Input	Handle for the channel.
inBufs	IVIDEO2_BufDesc *	Input	Input buffers for the channel.
outBufs	XDM2_BufDesc *	Output	Output buffers for the channel.
inArgs	IVIDENC2_InArgs *	Input	Input arguments for the channel.
outArgs	IVIDENC2_OutArgs *	Output	Output arguments for the channel.

4.2.2.15 IH264ENC_ProcessParamsList

|| Description

This structure defines the container for holding the N channel information.

|| Fields

Field	Data Type	Input/ Output	Description
numEntries	XDAS_Int32	Input	Number of channels in the given container.
enableErrorCheck	XDAS_Int32	Output	Checks the non supported features in N channel scenario

Field	Data Type	Input/ Output	Description
processParams []	IH264ENC_ProcessParams	Input	Array holding the process parameters. The array has a maximum of IH264ENC_MAX_LENGTH_PROCESS_LIST (24) elements.

4.2.2.16 IH264ENC_MetaDataFormatNaluInfo

|| Description

This structure defines the format of meta data used to provide information about slice.

|| Fields

Field	Data Type	Input/ Output	Description
`naluSize`	`XDAS_Int32`	Output	Size of each NAL Unit

4.2.2.17 IH264ENC_MetaDataFormatUserDefinedSEI

|| Description

This structure defines the format of meta data used to provide information about macro-block.

|| Fields

Field	Data Type	Input/ Output	Description
`Size`	`XDAS_Int32`	Input	Size of the payload
`payload[IH264ENC_MAX_SEI_METADTA_BUFSIZE]`	`XDAS_Int8`	Input	`Payload buffer holding the user defined SEI`

4.2.2.18 IH264ENC_Fxns

|| Description

This structure defines all the operations on H.264 Encoder instance objects.

|| Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
Ividenc	IVIDENC2_Fxns	Output	See IVIDENC2_Fxns data structure for details.
processMulti	XDAS_Int32 *fnPtr(IH264ENC_ProcessParamsList *processList)	Output	Function pointer to the multi-channel process call definition.

4.2.2.19 IH264ENC_VUICodingParams

|| Description

This structure contains all the parameters, which controls VUI parameters. Refer Annex E of the H.264 standard for more details of VUI and parameters

|| Fields

Field	Data Type	Input/ Output	Description
vuiCodingPreset	XDAS_Int8	Input	This preset controls the USER_DEFINED versus DEFAULT mode. If you are not aware about the fields, it should be set as IH264_VUICODING_DEFAULT
aspectRatioInfoPresentFlag	XDAS_UInt8	Input	This controls the insertion of aspect ratio information in VUI part of bit-stream ❑ zero : No aspect ratio related information is transmitted ❑ non-zero : aspect ratio related information is transmitted
aspectRatioIdc	XDAS_UInt8	Input	Encoder inserts aspectRatioIdc as it is in the bit-stream. It is user's responsibility to input appropriate value. See Table E-1 of H264 standard or enum IH264ENC_AspectRatioIdc for valid values. When aspectRatioIdc == IH264ENC_ASPECTRATIO_EXTENDED(255), 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. if aspectRatioInfoPresentFlag ==0 then encoder ignores this parameter

Field	Data Type	Input/ Output	Description
`videoSignalTypePresentFlag`	`XDAS_UInt8`	Input	This controls the insertion of video signal type in VUI part of bit-stream ❑ zero : No video signal related information is transmitted. ❑ non-zero : video signal related information is transmitted.
`videoFormat`	`XDAS_UInt8`	Input	This controls the video format type in VUI part of bit-stream. Encoder inserts `videoFormat`(lower 3 bits) as it is in the bit-stream. It is user's responsibility to provide appropriate value of this. See Table E-2 H264 standard or enum `IH264ENC_VideoFormat` for valid values.
`videoFullRangeFlag`	`XDAS_UInt8`	Input	This controls the video full range flag in VUI part of bit-stream. ❑ zero: video range is not full{0, 255} non-zero: video range is full
`timingInfoPresentFlag`	`XDAS_UInt8`	Input	This controls the insertion of timing info related parameters in VUI part of bit-stream ❑ zero: timing information is present ❑ non-zero: timing information is not present
`hrdParamsPresentFlag`	`XDAS_UInt8`	Input	This controls the insertion of HRD parameters in VUI part of bit-stream ❑ zero: HRD Parameters are present ❑ non-zero: HRD Parameters are not present
`numUnitsInTicks`	XDAS_UInt32	Input	This controls the insertion of `numUnitsInTicks` parameters in VUI part of bit-stream Valid values are [1, targetFrameRate] ❑ If this parameter is set by user then the `targetFrameRate` has multiplication factor of `numUnitInTicks` instead of 1000

4.2.2.20 IH264ENC_StereoInfoParams

‖ Description

This structure contains all the parameters, which controls stereo video coding. Refer Annex D of the H.264 standard for more details of Stereo Video Coding and parameters.

‖ Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
stereoInfoPreset	XDAS_UInt8	Input	This preset controls the Enable/Disable of stereo videoc coding. if enabled then `USER_DEFINED` or `DEFAULT` mode. If user wants stereo video coding and not aware about the fields, it should be set as `IH264_STEREO_ENABLE_DEFAULT`
topFieldIsLeftViewFlag	`XDAS_UInt8`	Input	This controls top field in video coded sequence as a left view or right view. ❑ zero : Top field is Left View ❑ non-zero: Top field is Right view
viewSelfContainedFlag	`XDAS_UInt8`	Input	This controls the Left/Right view should refer Left view or Right view. ❑ Zero ❑ Leftview can refer to Leftview or Rightview. ❑ Right view can refer to Leftview or Rightview. ❑ Non-zero ❑ Leftview can refer only to Leftview ❑ Rightview can refer only to Rightview

4.2.2.21 IH264ENC_FramePackingSEIParams

|| Description

This structure contains all the parameters, which controls Frame Packing SEI. Refer Annex D of the H.264 standard for more details of Frame Packing SEI Coding and parameters.

|| Fields

Field	Data Type	Input/ Output	Description
framePackingPreset	`XDAS_UInt8`	Input	This Preset controls the Enable/Disable of Frame packing SEI message encoding. If its enable then controls the `USER_DEFINED` vs `DEFAULT` mode. If User is not aware about following fields, it should be set as `IH264_FRAMEPACK_SEI_ENABLE_DEFAULT` ❑ 0: Frame packing SEI is Disabled (`IH264_FRAMEPACK_SEI_DISABLE`) ❑ 1: Default Frame packing SEI parameters (`IH264_FRAMEPACK_SEI_ENABLE_DEFAULT`) ❑ 2: User defined Frame packing SEI information pamameters (`IH264_FRAMEPACK_SEI_USERDEFINED`) When Frame packing SEI coding is enabled then input content type (coding type) should be Progressive coding.

Field	Data Type	Input/ Output	Description
framePackingType	`XDAS_UInt8`	Input	Indicates that frame packing arrangement type Refer IH264ENC_FramePackingType for possible values
frame0PositionX	`XDAS_UInt8`	Input	location of the upper left sample of frame 0 (Left view) in horizontal direction Note: Only the lower 4 bits are considered
frame0PositionY	`XDAS_UInt8`	Input	location of the upper left sample of frame 0 (Left view) in vertical direction Note: Only the lower 4 bits are considered
frame1PositionX	`XDAS_UInt8`	Input	location of the upper left sample of frame 1 (Right view) in horizontal direction Note: Only the lower 4 bits are considered
frame1PositionY	`XDAS_UInt8`	Input	location of the upper left sample of frame 1 (Right view) in vertical direction Note: Only the lower 4 bits are considered
reservedByte	`XDAS_UInt8`	Input	Value of `frame_packing_arrangement_reserved_byte` syntax element

4.2.2.22 IH264ENC_SVCCodingParams

|| Description

This structure contains all the parameters, which controls SVC Coding and parameters.

|| Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
svcExtensionFlag	XDAS_UInt8	Input	This parameter configures the codec to put SVC extensions in the bit-stream. For normal H.264 operation, this Flag needs to be ZERO (default value). For Encoder instance to encode SSPS, Prefix-NALU, Coded Slice in the bit-stream, this flag needs to be set. ❑ 0:IH264_SVC_EXTENSION_FLAG_DISABLE - Disables all SVC features/syntaxes and rest of the structure is not read/respected. ❑ 1:IH264_SVC_EXTENSION_FLAG_ENABLE - Encodes the required SVC related syntaxes of the layer for which H.264 Codec has been instantiated. ❑ 2:IH264_SVC_EXTENSION_FLAG_ENABLE_WITH_EC_FLEXIBILITY - Encodes the required SVC related syntaxes of the layer for which H.264 Codec has been instantiated. This mode is used to generate the bitstream which is compatible to TI-SVC decoder and which will make work easy of TI-SVC decoder's Error Concelment by putting info no_inter_layer_pred_flag in the svc-bitstream slice header.
dependencyID	XDAS_UInt8	Input	This parameter tell whether the current instance is for Base layer or for enhancement layer and also conveys Layer ID Info. This field is respected only when svcExtensionFlag is set. For configuring the encoder instance for BL then this parameter should be ZERO. For configuring the encoder instance for EL, this parameter should hold the value of the layer ID
qualityID	XDAS_UInt8	Input	This parameter tells Quality ID of the layer that the current instance of encoder is going to encode. This field is respected only when svcExtensionFlag is set. For configuring the encoder instance for BL then this parameter should be ZERO
enhancementProfileID	XDAS_UInt8	Input	This parameter conveys the enhancement encoder instance like what should be the profile ID to be encoded in the Sub-Sequence Parameter Set (SSPS).This parameter is dont care when, the svcExtensionFlag is not set.Possible values are IH264SVC_BASELINE_PROFILE (83) or IH264SVC_HIGH_PROFILE (86)
layerIndex	XDAS_UInt8	Input	This parameter conveys the enhancement encoder instance like what should be the pic_parameter_set_id and seq_parameter_set_id to be encoded in the Picture Parameter Set (PPS) and Sub-Sequence Parameter Set (SSPS). layerIndex is don't care or treated to be ZERO when svcExtensionFlag is not enabled

Field	Data Type	Input/ Output	Description
refLayerDQId	XDAS_Int8	Input	This parameter conveys the the DQ Id of the ReferenceLayer.

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
- IH264ENC_RateControlParams
- IH264ENC_InterCodingParams
- IH264ENC_IntraCodingParams
- IH264ENC_NALUControlParams
- IH264ENC_SliceCodingParams
- IH264ENC_LoopFilterParams
- IH264ENC_FMOCodingParams
- IH264ENC_VUICodingParams
- IH264ENC_StereoInfoParams
- IH264ENC_FramePackingSEIParams
- IH264ENC_SVCCodingParams
- IH264ENC_Params
- IH264ENC_DynamicParams

Table 4-5. Default and Supported Values for IVIDENC2_Params

Field	Default Value	Supported Value
Size	sizeof(IH264ENC_Params)	❑ sizeof(IVIDENC2_Params) ❑ sizeof(IH264ENC_Params)
ENCODINGPRESET	XDM_DEFAULT	❑ XDM_DEFAULT ❑ XDM_HIGH_SPEED ❑ XDM_USER_DEFINED ❑ XDM_HIGH_SPEED_MED_QUALITY [1] ❑ XDM_MED_SPEED_HIGH_QUALITY [2]
rateControlPreset	IVIDEO_STORAGE	❑ IVIDEO_STORAGE ❑ IVIDEO_NONE ❑ IVIDEO_USER_DEFINED ❑ IVIDEO_RATECONTROLPRESET_DEFAULT ❑ IVIDEO_LOW_DELAY
maxHeight	1088	[80, 4096] if contentType is IVIDEO_PROGRESSIVE [80, 2048]: if contentType is IVIDEO_INTERLACED
maxWidth	1920	[96, 4352]
dataEndianness	XDM_BYTE	XDM_BYTE

Field	Default Value	Supported Value
maxInterFrameInterval	1	[1,31] if contentType is IVIDEO_PROGRESSIVE [1, 16]: if contentType is IVIDEO_INTERLACED When(numTemporalLayer > 1), then maxInterFrameInterval = interFrameInterval = 1
maxBitRate	-1	Any Value, see Appendix N HFVBR .
minBitRate	0	Any Value, See Notes as part of section 4.2.1.7
inputChromaFormat	XDM_YUV_420SP	XDM_YUV_420SP
inputContentType	IVIDEO_PROGRESSIVE	❑ IVIDEO_PROGRESSIVE ❑ IVIDEO_PROGRESSIVE_FRAME ❑ IVIDEO_INTERLACED ❑ IVIDEO_INTERLACED_FRAME
operatingMode	IVIDEO_ENCODE_ONLY	IVIDEO_ENCODE_ONLY
Profile	IH264_HIGH_PROFILE	❑ IH264_BASELINE_PROFILE ❑ IH264_MAIN_PROFILE ❑ IH264_HIGH_PROFILE ❑ IVIDENC2_DEFAULTPROFILE
Level	IH264_LEVEL_40	❑ IH264_LEVEL_10 ❑ IH264_LEVEL_1b ❑ IH264_LEVEL_11 ❑ IH264_LEVEL_12 ❑ IH264_LEVEL_13 ❑ IH264_LEVEL_20 ❑ IH264_LEVEL_21 ❑ IH264_LEVEL_22 ❑ IH264_LEVEL_30 ❑ IH264_LEVEL_31 ❑ IH264_LEVEL_32 ❑ IH264_LEVEL_40 ❑ IH264_LEVEL_41 ❑ IH264_LEVEL_42 ❑ IH264_LEVEL_50 ❑ IH264_LEVEL_51 ❑ IVIDENC2_DEFAULTLEVEL
inputDataMode	IVIDEO_ENTIREFRAME	❑ IVIDEO_ENTIREFRAME ❑ IVIDEO_NUMROWS
outputDataMode	IVIDEO_ENTIREFRAME	❑ IVIDEO_ENTIREFRAME ❑ IVIDEO_FIXEDLENGTH ❑ IVIDEO_SLICEMODE ❑ When minBitRate != 0 then only IVIDEO_ENTIREFRAME is supported
numInputDataUnits	1	Ignored and assumed to be 1
numOutputDataUnits	1	[1,64]

Field	Default Value	Supported Value
metadataType[IVIDEO_MAX_NUM_METADATA_PLANES]	IVIDEO_METADATAPLANE_NONE	❑ IVIDEO_METADATAPLANE_NONE ❑ IH264_USER_DEFINED_SCALINGMATRIX ❑ IH264_SEI_USER_DATA_UNREGISTERED

Note:

- ❑ XDM_HIGH_SPEED_MED_QUALITY parameters are same as XDM_DEFAULT
- ❑ XDM_MED_SPEED_HIGH_QUALITY is same as XDM_DEFAULT except the change in minBlockSizeP and minBlockSizeB to IH264_BLOCKSIZE_8x8 instead of IH264_BLOCKSIZE_16x16
- ❑ For low delay rate control options maxInterFrameInterval can not be more than 1 and contentType can not be IVIDEO_INTERLACED

Table 4-6. Default and Supported Values for IVIDENC2_DynamicParams

Field	Default Value	Supported Value
size	sizeof(IH264ENC_DynamicParams)	❑ sizeof(IVIDENC2_DynamicParams) ❑ sizeof(IH264ENC_DynamicParams)
inputHeight	1088	[80, 4096] if contentType is IVIDEO_PROGRESSIVE [80, 2048]: if contentType is IVIDEO_INTERLACED
inputWidth	1920	[96, 4352]
refFrameRate	30000	Ignore
targetFrameRate	30000	Valid Values as per Level Limit
targetBitRate	12000000	Valid Values (> 16*1024) as per Level Limit
intraFrameInterval	30	Any value >=0
generateHeader	XDM_ENCODE_AU	XDM_ENCODE_AU XDM_GENERATE_HEADER
captureWidth	1920	>= inputWidth
forceFrame	IVIDEO_NA_FRAME	IVIDEO_NA_FRAME IVIDEO_IDR_FRAME
interFrameInterval	1	[1,31] if contentType is IVIDEO_PROGRESSIVE [1, 16]: if contentType is IVIDEO_INTERLACED
mvAccuracy	IVIDENC2_MOTIONVECTOR_QUARTERPEL	IVIDENC2_MOTIONVECTOR_QUARTERPEL IVIDENC2_MOTIONVECTOR_PIXEL

Field	Default Value	Supported Value
sampleAspectRatioHeight	1	Any value, only lower 16 bits are considered by encoder
sampleAspectRatioWidth	1	Any value, only lower 16 bits are considered by encoder
ignoreOutbufSizeFlag	XDAS_TRUE	[0,non-zero]
*putDataFxn	NULL	Valid function pointer, NULL
putDataHandle	0	Any Value
*getDataFxn	NULL	Valid function pointer, NULL
getDataHandle	0	Any Value
getBufferFxn	0	Valid function pointer, NULL
getBufferHandle	NULL	Valid function pointer, NULL
lateAcquireArg	IRES_HDVICP2_UNKNOWNLATEACQUIREARG (-1)	Any Value

Table 4-7. Default and Supported Values for IH264ENC_RateControlParams

Field	Default Value	Supported Value
rateControlParamsPreset	IH264_RATECONTROLPARAMS_DEFAULT	❑ IH264_RATECONTROLPARAMS_DEFAULT ❑ IH264_RATECONTROLPARAMS_USERDEFINED ❑ IH264 RATECONTROLPARAMS EXISTING
scalingMatrixPreset	IH264_SCALINGMATRIX_NORMAL(if profile == HIGH) IH264_SCALINGMATRIX_NONE(if profile != HIGH)	❑ IH264_SCALINGMATRIX_NONE ❑ IH264_SCALINGMATRIX_NORMAL ❑ IH264_SCALINGMATRIX_NOISY ❑ IH264_SCALINGMATRIX_STD_DEFAULT ❑ IH264_SCALINGMATRIX_USERDEFINED_SPSLEVEL ❑ IH264_SCALINGMATRIX_USERDEFINED_PPSLEVEL
rcAlgo	IH264_RATECONTROL_DEFAULT	❑ IH264_RATECONTROL_DEFAULT ❑ IH264_RATECONTROL_PRC ❑ IH264_RATECONTROL_PRC_LOW_DELAY
qpI	28	[-1,51]
qpMaxI	36	[0,51]
qpMinI	10	[0,51]
qpP	28	[-1,51]

Field	Default Value	Supported Value
qpMaxP	40	[0,51]
qpMinP	10	[0,51]
qpOffsetB	4	The value of (qpP + qpOffsetB) should be in range of [0,51]
qpMaxB	44	[0,51]
qpMinB	10	[0,51]
allowFrameSkip	0	Not supported – don't care
removeExpensiveCoeff	0	0,non-zero
chromaQPIndexOffset	0	[-12,12]
IPQualityFactor	IH264_QUALITY_FACTOR_DEFAULT	Ignore
initialBufferLevel	Equal to HRDBufferSize	❑ Any value between –(2^31 -10^8) to (2^31 -10^8)
HRDBufferSize	2*targetBitRate for VBR Rate Control ½*targetBitRate for CBR RateControl	Any value which is level compliant
minPicSizeRatioI	0	[0,4] & [5,31]
maxPicSizeRatioI	640	[0,960] except 31 and 32
minPicSizeRatioP	0	[0,4] & [5,31]
maxPicSizeRatioP	0	[0,960] except 31 and 32
minPicSizeRatioB	0	[0,4]&[5,31]
maxPicSizeRatioB	0	[0,960] except 31 and 32
enablePRC	1	[0, non-zero]
enablePartialFrameSkip	0	[0, non-zero]
discardSavedBits	0	[0, non-zero]
reserved	0	

Field	Default Value	Supported Value
VBRDuration	8	[0,3600]
VBRsensitivity	0	[0,8]
skipDistributionWindowLength	5	[0,10]
numSkipInDistributionWindow	1	[0,10]
enableHRDComplianceMode	1	[0, non-zero]
frameSkipThMulQ5	0	0 & [16,128]
vbvUseLevelThQ5	0	0 & [16,128]

Note:

For low delay rate control options maxInterFrameInterval can not be more than 1 and contentType can not be IVIDEO_INTERLACED

Table 4-8. Default and Supported Values for IH264ENC_InterCodingParams

Field	Default Value	Supported Value
interCodingPreset	IH264_INTERCODING_DEFAULT	❑ IH264_INTERCODING_DEFAULT ❑ IH264_INTERCODING_USERDEFINED ❑ IH264_INTERCODING_EXISTING ❑ IH264_INTERCODING_MED_SPEED_HIGH_QUALITY ❑ IH264_INTERCODING_HIGH_SPEED
searchRangeHorP	144	[16,144]
searchRangeVerP	32	[16,32]
searchRangeHorB	144	[16,144]
searchRangeVerB	16	16
interCodingBias	IH264_BIASFACTOR_DEFAULT	Ignore
skipMVCodingBias	IH264_BIASFACTOR_DEFAULT	❑ IH264_BIASFACTOR_DEFAULT ❑ IH264_BIASFACTOR_LOW ❑ IH264_BIASFACTOR_MILD ❑ IH264_BIASFACTOR_ADAPTIVE
minBlockSizeP	IH264_BLOCKSIZE_DEFAULT	❑ IH264_BLOCKSIZE_16x16 ❑ IH264_BLOCKSIZE_DEFAULT

Field	Default Value	Supported Value
		❑ IH264_BLOCKSIZE_8x8
minBlockSizeB	IH264_BLOCKSIZE_DEFAULT	❑ IH264_BLOCKSIZE_16x16 ❑ IH264_BLOCKSIZE_DEFAULT ❑ IH264_BLOCKSIZE_8x8
meAlgoMode	IH264ENC_MOTIONESTMODE_DEFAULT	❑ IH264ENC_MOTIONESTMODE_DEFAULT ❑ IH264ENC_MOTIONESTMODE_HIGH_SPEED

Note:

- minBlockSizeP and minBlockSizeB should be same if there is B frame.

Table 4-9. Default and Supported Values for IH264ENC_IntraCodingParams

Field	Default Value	Supported Value
intraCodingPreset	IH264_INTRACODING_DEFAULT	❑ IH264_INTRACODING_DEFAULT ❑ IH264_INTRACODING_USERDEFINED ❑ IH264_INTRACODING_EXISTING ❑ IH264_INTRACODING_HIGH_SPEED
lumaIntra4x4Enable	0xFF if (profile != IH264_HIGH_PROFILE) 0x0 if (profile == IH264_HIGH_PROFILE && inputContentType == IVIDEO_PROGRESSIVE) 0x1F if (profile == IH264_HIGH_PROFILE && inputContentType != IVIDEO_PROGRESSIVE)	[0x000, 0x1FF]
lumaIntra8x8Enable	0x0 if (profile != IH264_HIGH_PROFILE) 0xFF if (profile == IH264_HIGH_PROFIL	[0x000, 0x1FF]

Field	Default Value	Supported Value
	E && inputContentType == IVIDEO_PROGRESSIVE) 0x1F if (profile == IH264_HIGH_PROFILE && inputContentType != IVIDEO_PROGRESSIVE)	
lumaIntra16x16Enable	0xF	[0x0, 0xF]
chromaIntra8x8Enable	0xF	[0x0, 0xF]
chromaComponentEnable	IH264_CHROMA_COMPONENT_DEFAULT	❑ IH264_CHROMA_COMPONENT_CR_ONLY ❑ IH264_CHROMA_COMPONENT_CB_CR_BOTH
intraRefreshMethod	IH264_INTRAREFRESH_DEFAULT	❑ IH264_INTRAREFRESH_DEFAULT ❑ IH264_INTRAREFRESH_CYCLIC_MBS ❑ IH264_INTRAREFRESH_GDR
intraRefreshRate	0	>=0, effective only intraRefreshMethod != IH264_INTRAREFRESH_DEFAULT
gdrOverlapRowsBtwFrames	0	<= intraRefreshRate
constrainedIntraPredEnable	0	Zero, non-zero
intraCodingBias	IH264ENC_INTRACODINGBIAS_DEFAULT	❑ IH264ENC_INTRACODINGBIAS_DEFAULT ❑ IH264ENC_INTRACODINGBIAS_HIGH_SPEED

Table 4-10. Default and Supported Values for IH264ENC_NALUControlParams

Field	Default Value	Supported Value
naluControlPreset	IH264_NALU_CONTROL_DEFAULT	❑ IH264_NALU_CONTROL_DEFAULT ❑ IH264_NALU_CONTROL_USERDEFINED
naluPresentMaskStartOfSequence	0x01A0	See Appendix B for more details
naluPresentMaskIDRPicture	0x01A0	See Appendix B for more details

Field	Default Value	Supported Value
naluPresentMaskIntraPicture	0x0002	See Appendix B for more details
naluPresentMaskNonIntraPicture	0x0002	See Appendix B for more details
naluPresentMaskEndOfSequence ;	0x0C00	See Appendix B for more details

Table 4-11. Default and Supported Values for IH264ENC_SliceCodingParams

Field	Default Value	Supported Value
sliceCodingPreset	IH264_SLICECODING_DEFAULT	❑ IH264_SLICECODING_DEFAULT ❑ IH264_SLICECODING_USERDEFINED ❑ IH264_SLICECODING_EXISTING
sliceMode	IH264_SLICEMODE_DEFAULT	❑ IH264_SLICEMODE_NONE ❑ IH264_SLICEMODE_MBUNIT ❑ IH264_SLICEMODE_OFFSET ❑ IH264_SLICEMODE_BYTES
sliceUnitSize	0	[6,number_of_mbs_in_picture]: when sliceMode == IH264_SLICEMODE_MBUNIT [256, Any Number]: when sliceMode == IH264_SLICEMODE_BYTES Ignore if sliceMode != IH264_SLICEMODE_MBUNIT && sliceMode != IH264_SLICEMODE_BYTES
sliceStartOffset[IH264ENC_MAX_NUM_SLICE_START_OFFSET]	{0, 0, 0}	Increasing order: Any Value >=0 .
streamFormat	IH264_STREAM_FORMAT_DEFAULT	IH264_BYTE_STREAM IH264_NALU_STREAM

Note:

- sliceMode == IH264_SLICEMODE_BYTES is only supported under below conditions:

  Width >= 128 pixels

  inputContentType != IVIDEO_INTERLACED

  entropyCodingMode != IH264_ENTROPYCODING_CABAC

  interFrameInterval == 1 (No 'B' Frames)

  streamFormat==IH264_NALU_STREAM is only supported when outputDataMode == IVIDEO_SLICEMODE with sub frame level communications

Table 4-12. Default and Supported Values for IH264ENC_LoopFilterParams

Field	Default Value	Supported Value
loopfilterPreset	IH264_LOOPFILTER_DEFAULT	❑ IH264_LOOPFILTER_DEFAULT ❑ IH264_LOOPFILTER_USERDEFINED
loopfilterDisableIDC	IH264_DISABLE_FILTER_DEFAULT	❑ IH264_DISABLE_FILTER_NONE ❑ IH264_DISABLE_FILTER_ALL_EDGES ❑ IH264_DISABLE_FILTER_SLICE_EDGES
filterOffsetA	0	[-12, 12] even value
filterOffsetB	0	[-12, 12] even value

Table 4-13. Default and Supported Values for IH264ENC_FMOCodingParams

Field	Default Value	Supported Value
fmoCodingPreset	IH264_FMOCODING_DEFAULT	IH264_FMOCODING_NONE
numSliceGroups	1	Ignore
sliceGroupMapType	IH264_SLICE_GRP_MAP_DEFAULT	Ignore
sliceGroupChangeDirectionFlag	IH264ENC_SLICEGROUP_CHANGE_DIRECTION_DEFAULT	Ignore
sliceGroupChangeRate	0	Ignore
sliceGroupChangeCycle	0	Ignore
sliceGroupParams[MAXNUMSLCGPS]	{0 ,0}	Ignore

Table 4-14. Default and Supported Values for IH264ENC_VUICodingParams

Field	Default Value	Supported Value
vuiCodingPreset	IH264_VUICODING_DEFAULT	IH264_VUICODING_DEFAULT IH264_VUICODING_USERDEFINED
aspectRatioInfoPresentFlag	0	0,non-zero
aspectRatioIdc	0	[0,255]: No Error Check, user is responsible to provide correct value
videoSignalTypePresentFlag	0	0,non-zero

Field	Default Value	Supported Value
videoFormat	IH264ENC_VIDEOFORMAT_NTSC	❑ IH264ENC_VIDEOFORMAT_COMPONENT ❑ IH264ENC_VIDEOFORMAT_PAL ❑ IH264ENC_VIDEOFORMAT_NTSC ❑ IH264ENC_VIDEOFORMAT_SECAM ❑ IH264ENC_VIDEOFORMAT_MAC ❑ IH264ENC_VIDEOFORMAT_UNSPECIFIED
videoFullRangeFlag	0	0,non-zero
timingInfoPresentFlag	0	0,non-zero
hrdParamsPresentFlag	0	0,non-zero
numUnitsInTicks	1000	[1,targetFrameRate]

Table 4-15. Default and Supported Values for IH264ENC_StereoInfoParams

Field	Default Value	Supported Value
stereoInfoPreset	IH264_STEREOINFO_DISABLE	❑ IH264_STEREOINFO_DISABLE ❑ IH264_STEREOINFO_ENABLE_DEFAULT ❑ IH264_STEREOINFO_ENABLE_USERDEFINED
topFieldIsLeftViewFlag	1	0,non-zero
viewSelfContainedFlag	0	0,non-zero

Table 4-16. Default and Supported Values for IH264ENC_FramePackingSEIParams

Field	Default Value	Supported Value
framePackingPreset	IH264_FRAMEPACK_SEI_DISABLE	❑ IH264_FRAMEPACK_SEI_DISABLE ❑ IH264_FRAMEPACK_SEI_ENABLE_DEFAULT ❑ IH264_FRAMEPACK_SEI_USERDEFINED
framePackingType	IH264_FRAMEPACK_TYPE_DEFAULT	❑ IH264_FRAMEPACK_CHECKERBOARD ❑ IH264_FRAMEPACK_COLUMN_INTERLEAVING ❑ IH264_FRAMEPACK_ROW_INTERLEAVING ❑ IH264_FRAMEPACK_SIDE_BY_SIDE ❑ IH264_FRAMEPACK_TOP_BOTTOM
frame0PositionX	0	[0,15]
frame0PositionY	0	[0,15]
Frame1PositionX	0	[0,15]

Field	Default Value	Supported Value
Frame1PositionY	0	[0,15]
reservedByte	0	[0,255]

Table 4-17. Default and Supported Values for IH264ENC_SVCCodingParams

Field	Default Value	Supported Value
svcExtensionFlag	IH264_SVC_EXTENSION_FLAG_DISABLE	❑ IH264_SVC_EXTENSION_FLAG_DISABLE ❑ IH264_SVC_EXTENSION_FLAG_ENABLE ❑ IH264_SVC_EXTENSION_FLAG_ENABLE_WITH_EC_FLEXIBILITY
dependencyID	0	[0,255]
qualityID	0	[0,255]
enhancementProfileID	0	[0,255]
layerIndex	0	[0,255]
refLayerDQId	0	[0,255]

Table 4-18. Default and Supported Values for IH264ENC_Params

Field	Default Value	Supported Value
videnc2Params	See Table 4-5. Default and Supported Values for IVIDENC2_Params	
rateControlParams	See Table 4-7. Default and Supported Values for IH264ENC_RateControlParams	
interCodingParams	See Table 4-8. Default and Supported Values for IH264ENC_InterCodingParams	
intraCodingParams	See Table 4 9. Default and Supported Values for IH264ENC_IntraCodingParams	
nalUnitControlParams	See Table 4-10. Default and Supported Values for IH264ENC_NALUControlParams	
sliceCodingParams	See Table 4-11. Default and Supported Values for IH264ENC_SliceCodingParams	
loopFilterParams	See Table 4-12. Default and Supported Values for IH264ENC_LoopFilterParams	
fmoCodingParams	See Table 4-13. Default and Supported Values for IH264ENC_FMOCodingParams	
vuiCodingParams	See Table 4-14. Default and Supported Values for IH264ENC_VUICodingParams	

Field	Default Value	Supported Value
stereoInfoParams	See Table 4-15. Default and Supported Values for IH264ENC_StereoInfoParams	
framePackingSEIParams	See Table 4-16. Default and Supported Values for IH264ENC_FramePackingSEIParams	
svcCodingParams	See Table 4 17. Default and Supported Values for IH264ENC_SVCCodingParams	
interlaceCodingType	IH264_INTERLACE_FIELDONLY_ARF	❑ IH264_INTERLACE_FIELDONLY ❑ IH264_INTERLACE_FIELDONLY_MRF ❑ IH264_INTERLACE_FIELDONLY_ARF ❑ IH264_INTERLACE_DEFAULT ❑ IH264_INTERLACE_FIELDONLY_SPF
bottomFieldIntra	0	0, non-zero
gopStructure	IH264ENC_GOPSTRUCTURE_NONUNIFORM	❑ IH264ENC_GOPSTRUCTURE_NONUNIFORM ❑ IH264ENC_GOPSTRUCTURE_DEFAULT ❑ IH264ENC_GOPSTRUCTURE_UNIFORM
entropyCodingMode	IH264_ENTROPYCODING_CAVLC(if Profile == BASELINE) IH264_ENTROPYCODING_CABAC(if Profile != BASELINE)	❑ IH264_ENTROPYCODING_CABAC ❑ IH264_ENTROPYCODING_CAVLC
transformBlockSize	IH264_TRANSFORM_ADAPTIVE (if Profile == HIGH) IH264_TRANSFORM_4x4 (if Profile != HIGH)	❑ IH264_TRANSFORM_4x4 ❑ IH264_TRANSFORM_8x8 ❑ IH264_TRANSFORM_ADAPTIVE
log2MaxFNumMinus4	10	[0,12]
picOrderCountType	IH264_POC_TYPE_0	❑ IH264_POC_TYPE_0 ❑ IH264_POC_TYPE_1 ❑ IH264_POC_TYPE_2
enableWatermark	0	[0, non-zero]
IDRFrameInterval	0	Any value
pConstantMemory	NULL	NULL, Valid Address pointing to constants in DDR
maxIntraFrameInterval	1	Any value >= 0
debugTraceLevel	0	Zero, non-zero (all non-zero values are considered as same level)
lastNFramesToLog	0	Any Value

Field	Default Value	Supported Value
enableAnalyticinfo	0	Zero, non-zero
enableGMVSei	0	Zero, non-zero
constraintSetFlags	0	Zero, non-zero
enableRCDO	0	Zero, non-zero
enableLongTermRefFrame	IH264ENC_LTRP_NONE	❑ IH264ENC_LTRP_NONE ❑ IH264ENC_LTRP_REFERTO_PERIODICLTRP ❑ IH264ENC_LTRP_REFERTOP_PROACTIVE ❑ IH264ENC_LTRP_REFERTOP_REACTIVE enableLongTermRefFrame should be IH264ENC_LTRP_NONE, when interFrameInterval > 1
LTRPPeriod	0	Zero, nonZero positive value When numTemporalLayer > 1, it should be multiple of Sub-Gop length
numTemporalLayer	❑ IH264_TEMPORAL_LAYERS_1	❑ [IH264_TEMPORAL_LAYERS_1 , IH264_TEMPORAL_LAYERS_4]
referencePicMarking	❑ IH264_LONG_TERM_PICTURE	❑ IH264_SHORT_TERM_PICTURE ❑ IH264_LONG_TERM_PICTURE
reservedParams[3]	0,0,0,0	Ignore

Table 4-19. Default and Supported Values for IH264ENC_DynamicParams

Field	Default Value	Supported Value
videnc2DynamicParams	See Table 4-6. Default and Supported Values for IVIDENC2_DynamicParams	
rateControlParams	See Table 4-7. Default and Supported Values for IH264ENC_RateControlParams	
interCodingParams	See Table 4-8. Default and Supported Values for IH264ENC_InterCodingParams	
intraCodingParams	See Table 4-8. Default and Supported Values for IH264ENC_InterCodingParams	
sliceCodingParams	See Table 4-11. Default and Supported Values for IH264ENC_SliceCodingParams	
sliceGroupChangeCycle	0	Ignore
searchCenter	{0x7FFF,0x7FFF}	(-64,64), 0x7FFF --> ignore user provided gMV and use internal
enableStaticMBCount	0	Zero, non-zero
enableROI	0	Zero, non-zero
reservedDynParams[3]	0	Ignore

4.4 Interface Functions

This section describes the Application Programming Interfaces (APIs) used in the H.264 Encoder. The APIs are logically grouped into the following categories:

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

You must call these APIs in the following sequence:

1) `algNumAlloc()`
2) `algAlloc()`
3) `algInit()`
4) `algActivate()`
5) `process()/processMulti()`
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 section 4.2 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).

Since there is no mechanism to return extended error code for unsupported parameters, this version of encoder returns `IALG_EOK` even if some parameter unsupported is set. But subsequence control/process call it returns the detailed error code

|| 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 section 4.2 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 */

XDM_EUNSUPPORTED; /* status indicating parameters not
supported*/
```

|| 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 status or handle is NULL then codec returns `IALG_EFAIL`
- If the control command is not recognized or some parameters to act upon are not supported, the return value from this operation is not equal to `XDM_EUNSUPPORTED`.
- 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 `IVIDEO2_BufDesc` and `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 `XDM_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 H.264 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

processMulti() – N channel video encoding call

‖ Synopsis

```
XDAS_Int32 (*processMulti) (IH264ENC_ProcessParamsList *processList);
```

‖ Arguments

```
IH264ENC_ProcessParamsList *processList; /* Container for N channels.
Each channel structure(IH264ENC_ProcessParams) contains handle, *inBufs,
*outBufs, *inArgs, *outArgs */
```

‖ Return Value

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

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

‖ Description

This function does the basic H264 video encoding for N channels. The argument to processMulti() is a container for N channels. The structure `IH264ENC_ProcessParams` contains five parameters. The first parameter is a handle to an algorithm instance.

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

The fourth parameter is a pointer to the `IVIDENC2_InArgs` data structure that defines the run time input arguments for an algorithm instance object.

The last parameter 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.

- `processMulti()` can only be called after a successful return from `algInit()` and `algActivate()`.
- If algorithm uses DMA resources, `processMulti()` 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 `XDM_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 `processMulti()` 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 H.264 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()`

This page is intentionally left blank

Chapter 5

Frequently Asked Questions

This chapter provides answers to few frequently asked questions related to using this encoder.

5.1 Release Package

Question	Answer
Can this codec release be used on any HDVICP2 and Media Controller based platform?	Yes, you can use it on any HDVICP2 and Media Controller based platforms (eg DM816x, DM814x). 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/data memory section is not sufficient for placement	Make sure that project settings are not changed from the released package. Change in debug options for compilation may make code/data memory size insufficient for placement.
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.

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.)
Which simulator version should I use for this release?	Code Composer Studio (CCSv4) version 4.2.0.09000 has to be installed. Netra 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

Question	Answer
What CG tools version is used for this release?	CG tools version 4.5.1 is used for this release.
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.
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

5.4 Algorithm Related

Question	Answer
What XDM interface does codec support?	Codec supports XDM IVIDENC2 interface
What are the profiles supported in this version of encoder?	This version of encoder supports baseline, main and high profiles. FMO feature is not supported for baseline profile.
What is the maximum level supported by this encoder?	The encoder supports the level up to 5.1
What is the maximum bit rate supported?	Maximum bit rate depends upon the level setting. This version supports the maximum bit rate of 50 mbps (Base Line and Main profile Level 4.2) and 62.5 mbps (High Profile Level 4.2) for 30 fps case. To achieve the real time performance with CABAC bit rate should be less than 25 Mbps for 30fps case
Can I encode with bit rate or any other parameter (example resolution) more than specified in level 4.2?	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 encode with level higher than 4.2	Yes. Functional point of view encoder support Level 5.1 (with contarsints on resolution not be higher than 4352x4096). But performance is not guaranteed to be real time
Can I reduce DDR footprint of encoder?	Yes. DDR foot print is majorly dependent on maxWidth and maxHeight parameters and also dependent on whether long term reference frame is enabled or not.
What stream formats are supported in this version of encoder?	This version supports byte-stream and NALU format
What are the input frame formats supported? Can I encode YUV 422 input format buffer?	This version supports only YUV420 semi-planar input format only. No. other formats than YUV420 semi-planar are not supported

Question	Answer
What is granularity of the process call?	The encoder supports only frame level encoding API. However, it also supports data sync APIs for output bit stream, which is a call back to the application for data synchronization.
What are the resolutions supported?	The encoder supports all resolutions until up to 4352x4096. The minimum resolution supported is 96x80. Width has to be multiple of 16 but height can be any number
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 ❑ 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 `ignoreOutBufSizeFlag = 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
Will change in above parameters result in a IDR insertion	Change in resolution will result in IDR insertion. Change in bit rate may insert IDR if HRD parameters are coded as part of bit-stream. Similarly if timing info related parameters are coded in bit-stream then it can cause insertion of IDR by doing change in frame rate
Does the encoder support B frame encoding? In what order does encoder expect the frames, encode order or capture order? How the delay is controlled?	Yes, encoder supports B frame encoding. It accepts the frames in capture order and internally processes them in encoder order. Encoder has a mechanism to lock and free the input buffer, based on this it has a initial delay to produce the bit stream, which is equivalent to number of B frames getting encoded. Subsequent process call should produce the compressed bit-stream and also frees up a buffer.
How many continuous B frames can I have? Is there a performance/quality impact?	In case of progressive content maximum 31 continuous B frames can be produced. With interlaced content maximum 32 B fields can be produced Quality is not tuned for more than two B frames so for motion sequences it is not advised to have more than two B frames Performance is impacted slightly; this is because if B frames are more than two then some information related to buffers are stored in external memory compared to internal memory because of limited DTCM. Hence, it affects the performance.
Does the encoder support meta data input/output?	Yes, this version of the encoder supports reading in meta data for user data unregistered SEI and user defined scaling matrix. For more details on how this data is written See A and C.
Does this version of H264 Encoder expose motion vectors for a frame to the application?	Yes
Can encoder take the motion vectors given externally for encoding or say in a transcode scenario?	No

Question	Answer
Can codec do frame rate conversion?	No, refFrameRate and targetFrameRate needs to be same.
Does this version of encoder support interlaced coding?	Yes, this version of H.264 Encoder supports interlaced coding with field only coding. MBAFF and PICAFF are not supported. However, controls to decide parity of reference field are given to user, like SPF, MRF, ARF.
In case of interlaced, will single encode (Process call), encode both the fields?	Encoder allows both fields processing in single process call as well process call per field.
Does Algorithm support sub-frame level communication mechanism for low-delay applications?	Yes. It has the mechanism for sub-frame level communication for both input and output buffers.
Does this version of encoder support encoding multiple slices in a frame?	Yes, slices can be generated bases upon number of macro blocks per slice, number of bytes per slice and also based upon the row start offset in a frame
Is there a limit on number of slices supported per frame by encoder?	Yes, encoder can generate one slice per 6 macroblocks not below that when configured in sliceMode = IH264_SLICEMODE_MBS. When sliceMode = IH264_SLICEMODE_MBS, it can allow minimum value of bytes per slice as 256
Does Algorithm support H.241 based packetization (slice cap/ maxBitsPerSlice) feature ?	Yes.
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 to 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).
In case of interlaced, can bottom field come first in bit stream?	Yes. A sequence can look like this: BF, TF, BF, TF, BF, TF…. Encoder allows accepting the information as top field is first field or not
For Interlace content how the YUV data are expected, is it interleaved or field separated	Both format is supported, interleaved and field separated.
Can frac-pel refinement of motion vectors be disabled?	Yes
Can the encoder give multiple Motion vector for a macro block?	Yes
Is there any performance difference between 1MV and 4MV per macro block?	Yes, please refer the data sheet for the impact on performance

Question	**Answer**
What is the behavior of Codec on cache properties of input and output buffer	All input and output buffer of encoder are read/written by DMA. So codec assumes that all input data is valid in DDR memory before feeding in to encoder. Also outout of encoder is guaranteed to be in DDR. Hecne the parameters like `InBufs : IVIDEO2_BufDesc.planeDesc[idx].usageMode and OuBufs : XDM2_BufDesc.Descs[0].usageMode` are don't care However for the trace and debug related buffers produced by encoder it is not true. There are some buffers for which data can be in cache memory and cache write back from application side will be needed, refer Appendix E for more details
What is `rateControlPreset` and `rateControlParamsPreset` ? What is the difference between these two?	`rateControlPreset` control the rate control algorithm (`IH264ENC_RateControlParams ::rcAlgo`) but `rateControlParamsPreset` controls the other associated parameters specified in `IH264ENC_RateControlParams` structure. When `rateControlPreset` is user defined then only `IH264ENC_RateControlParams ::rcAlgo` is resepected otherwise it is controlled by `rateControlPreset`. But even if `rateControlPreset` is not user defined other parameters of `IH264ENC_RateControlParams` structure are possible to be user controllable by setting `rateControlParamsPreset` as user defined
Does the encoder support multi-channel operation?	Yes.
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.
Does a Luma buffer and corresponding Chroma buffer needs to be contiguous in memory?	No
Can the encoder generate headers only?	Yes, have a control call of XDM_GENERATE_HEADER before the particular process call.
Does encoder support skipping of frames?	Yes, encoder will encode requested frame as all macro block as skipped MB. Please refer to user guide for further details
What is the benefit of asking a frame as skip	It can help to balance the performance or bitrate in certain situations. The frame being asked to be coded as skip consumes very less MHZ of the HDVICP2. It can finish the entire frame/field processing in less than 5 MHz
Is it possible to configure the stream format (Byte stream vs NAL stream format) at frame level run-time?	No, it can be configured only at create time
How to use interlaced encoding in H.264 encoder	`You need to configure contentType as IVIDEO_INTERLACED and provide the pointers to field buffers appropriatelty during process call`

Question	Answer
How to change resolution dynamically?	You need to make a control call of encoder with `XDM_SETPARAMS` command. At this time configure the inputWidth and inputHeight parameter indicating the new resolution. Subsequnt process call we start assuming the newly configured resolution.
How to change frame rate, bitrate or any other dynamic parameter dynamically?	You need to make a control call of encoder with `XDM_SETPARAMS` command. At this time configure the appropriate parameter with new value. Subsequnt process call we start assuming the newly configured resolution.
How to force Intra frames in H.264 encoder	You need to make a control call of encoder with `XDM_SETPARAMS` command and forceFrame = IVIDEO_IDR_FRAME. Subsequnt process call will be coded as IDR frame. The effect of this control call is only for one frame and subsequent frame will be coded as per defined gop structure
How to generate SPS and PPS headers in bit-stream?	Refer Appendix B. If you want dynamically before certain frames then use `XDM_GENERATE_HEADER`
How to insert user data SEI message in H.264 bitstream	Refer Appendix A
How to insert picture timing SEI message?	Refer Appendix B
What is the latency of the codec?	This encoder is designed for low latency applications hence it can take uncompressed data with a minimum unit of 1 MB row (16 lines) and can provide compressed bit-stream out with a minumum unit of 1 slice (compressed unit used for packets). Now based upon what is the slice rate - one can compute the latency at which compressed data will be available at encoder output Example - assume each frame has 20 slices then each slice is available at the output of the encoder at (33 ms / 20 + 0.3 ms intial overhead) time interval =~ 2 ms So you should be able to compute the latency for you application based upon slice rate.
Can H.264 encoder do all Intra frames as IDR encoding?	Yes.
Can H.264 encoder do all Intra frames encoding?	Yes, H.264 encoder can do all intra frames encoding. One has to set IntraFrameInterval with correct value. If you want to reduce DDR foot print for this use case then configure create time parameter `maxIntraFrameInterval = 1`
How many channels of H264 Encoder can be supported?	Given the standalone data for each resolution in data sheet, please do the math yourself accounting for the MHz clock of HDVICP2 and DDR Bandwidth on the SoC.
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**
How will the application know when to stop calling process function after applying XDM_FLUSH?	When application puts the encoder in flush mode by calling control with XDM_FLUSH, encoder starts encoding locked frames if any. Application needs to call process in a do-while loop till the encoder return XDM_INSUFFICIENTDATA error.
How to enable Hierarchical P – coding?	Configure the parameter `numTemporalLayer` to a value greater than 1. Current version of encoder supports upto a maximum of 4 temporal layers.
How to enable Watermark feature in Enocder?	It can be enabled by setting any non-zero value to the parameter `enableWatermark` as a part of IH264ENC_Params . And the key is passed to encoder through inputKey as a part of IH264ENC_InArgs.
How many watermark SEI messages are inserted in the stream for interlace case?	Only one SEI message is inserted for a pair of fields
Do we need to pass two input keys(one key per field) in interlaced coding?	No. Enocder accepts only one key for a pair of fields in interlaced coding. In case of 60 process call encoding, the key fed for the second field is considered. For more details, refer 'Appendix L'.
Does encoder generates watermark SEI message when a process call is with made XDM_GENERATE_HEADER enabled?	No. There will not be any SEI message for this scenario. And the input Key passed to encoder in this process call is ignored.
Does encoder supports all the features for resolution more than 2kx2k?	Yes.
Suppose the user has configured MaxWidth/Maxheight more than 2048x2048 and actual encoding inputWidth/inputHeight are less than 2048x2048. Any precautions or features unsupport for this configuration?	Yes, there are few suggestions to the user in this configuration, 1. There would be performance degradtion of about 12 to 15MHz for 30 frames HD resolution. 2.If it is a prior known that the dynamic change in resolution never goes beyond 2048x2048, then always configure maxWidthxmaxHeight to 2048x2048 for better performance.
Can the user give separate ROI Input parameters for each field in Interlaced Cases?	No. Both the fields will use the same ROI Input parameters. For more details, refer 'Appendix K'.
Any performance degradation with ROI enabled?	Yes, Around 4MHz overhead for 30 frames encoding of 1080p resolution input with 1MV and around 10MHz for 30 frames encoding of 1080p resolution streams with 4MV enabled. n streams with 4MV enabled. Performance also depends on the number of Roi regions in a frame.
How can user specify the colour for privacy masked region?	Set `IH264ENC_RoiInput->roiPriority`[] with the Y,Cb,Cr combination. The data type of this parameter is a integer(4 bytes) array. 0th byte = Cr , 1st byte = Cb , 2nd byte = Y and 3rd byte is ignored. By default all these values are zero(grey colour).

5.5 Trouble Shooting

Question	Answer
Encoder generates an output bit stream which has garbage frames?	Please check whether the input YUV given to encoder is proper or not. Encoder expects/supports the YUV NV12 format only.
In the encoded bit stream luma information looks proper but not chroma information.	Please check the input YUV format fed into encoder. Encoder supports only YUV NV12 format.
Codec misbehaves or hangs when sliceMode = IH264_SLICEMODE_BYTES	This is a known shortcoming in the simulator. This feature has been verified on hardware.
In the first process call, I am getting the error as IH264VDEC_ERR_HDVICP2_IMPROPER_STATE	Before HDVICP2 is given to codec, HDVICP2 has to be in standby mode. Other wise this error will show up. So check the HDVICP2_Reset functionality used in the Application side. For sample flow and implementation, refer Test Application in the release package. Note that in some configurations of simulator, reset might not be needed.
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.
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 ❑ Run time error occurred during encoding of the frame

Appendix A

Meta Data Support

This appendix explains the meta data support by encoder. Encoder supports multiple meta data as consumer as well as producer.

A.1 Control Parameter to Enable/Disable Metadata

This feature can be enabled/disabled through create time parameters `IVIDENC2_Params::metadataType[IVIDEO_MAX_NUM_METADATA_PLANES]`. There can be maximum `3(IVIDEO_MAX_NUM_METADATA_PLANES)` meta data planes possible to be supported with one instance of encoder.

Each element of `metadataType[]` array can possibly take following enumerated values. For supported values with this version of encoder, please refer Table 4.5.

Enumeration	Value
IVIDEO_METADATAPLANE_NONE	-1
IVIDEO_METADATAPLANE_MBINFO	0
IVIDEO_METADATAPLANE_EINFO	1
IVIDEO_METADATAPLANE_ALPHA	2
IH264_SEI_USER_DATA_UNREGISTERED	256
IH264_REGION_OF_INTEREST	257
IH264_USER_DEFINED_SCALINGMATRIX	258

If user wants to pass user defined scaling matrix via meta data plane 2 then `IVIDENC2_Params::metadataType[2]` should be set to `IH264_USER_DEFINED_SCALINGMATRIX`.

If user don't want to use any meta data plane then all the entries of `IVIDENC2_Params::metadataType[]` should be set to `IVIDEO_METADATAPLANE_NONE`

A.2 Format of meta data

Format of Each meta data that is supported has to be defined by the encoder. The format for each supported meta data is explained below:

A.2.1 SEI_USER_DATA_UNREGISTERED

For this purpose encoder allows only one meta data (not multiple units of this meta data). The format is as shown below:

Size (32-bit)	Payload[size]

The maximum value of size can be 1023 bytes. Encoder only reads the lower 10-bits of the size field

A.2.2 MBINFO

Format of this meta data is yet to be defined

A.2.3 ROI

Please refer Appendix K for the details related to format of this meta data

A.2.4 USER_DEFINED_SCALINGMATRIX

Please refer Appendix C for the details related to format of this meta data.

A.3 Steps to enable a meta data with Example

The way to pass meta data to encode is through `inBufs` to the encoder during process call. The way to get meta data from encode is through `outBufs` of the encoder during process call.

When application request the buffer information through control call with `XDM_GETBUFINFO`, encoder considers `IVIDENC2_Params::metadataType[]` array to count the buffers required at input/output level. For each meta data one additional buffer is required. If for some metadata size is not known by encoder then it should return size =-1 so that application can allocate as per its knowledge. Same way for some meta-data application might not provide the size to codec through `XDM2_SingleBufDesc.bufSize.bytes`, in that case application can set it to -1. The meta data which has size set to -1 should have first word (32-bit) of meta data as size and properly updated.

For Example: User want to insert `SEI_USER_DATA_UNREGISTERED` meta data at each IDR picture, the following steps should be followed

1) Create the encoder object with IVIDENC2_Params::metadataType[IDX_SEI_METADATA] = IH264_SEI_USER_DATA_UNREGISTERED

2) Also the user data un-registered SEI bit in the NAL unit mask for IDR picture should be set IH264ENC_SET_NALU(naluPresentMaskIDRPicture, USER_DATA_UNREGD_SEI)

3) Call Control function with XDM_GETBUFINFO. Encoder should return one additional input buffer as required, size of the buffer will be -1 as encoder doesn't know the size

Application should have a memory allocated for this meta data and pass on to the encoder via

As mentioned in section A.2.1 this meta-data format includes size field, so encoder will read size from the actual meta data buffer and utilize the buffer.

```
IVIDEO2 BufDesc *inBufs->numMetaPlanes = 1 ;

inBufs->metadataPlaneDesc[IDX SEI METADATA].buf = pBuffer ;

inBufs->metadataPlaneDesc[IDX_SEI_METADATA].bufSize.bytes = -1
```

`ppBuffer` points to this buffer in memory

Size (32-bit)	Payload[size]

Here `IDX_SE_METADATA` can be any value 0 to 2 (`IVIDEO_MAX_NUM_METADATA_PLANES-1`).

This page is intentionally left blank

Appendix B

Control for Configurable NALU

This appendix explains the configurable NAL unit support by encoder. This is to help the application to decide the position of few key NAL units at different position in video sequence

B.1 Position in Video Sequence

There are five main positions in a video sequence

Start of the Sequence

I frame

IDR frame

End of Sequence

All other positions which are not identified by above 4 positions (non Intra frame positions)

B.2 NAL Units in H.264 Video Sequence

There are following possible NAL units in H.264 encoder.

1) `IH264_NALU_TYPE_UNSPECIFIED`
2) `IH264_NALU_TYPE_SLICE`
3) `IH264_NALU_TYPE_SLICE_DP_A`
4) `IH264_NALU_TYPE_SLICE_DP_B`
5) `IH264_NALU_TYPE_SLICE_DP_C`
6) `IH264_NALU_TYPE_IDR_SLICE`
7) `IH264_NALU_TYPE_SEI`
8) `IH264_NALU_TYPE_SPS`
9) `IH264_NALU_TYPE_PPS`
10) `IH264_NALU_TYPE_AUD`
11) `IH264_NALU_TYPE_EOSEQ`
12) `IH264_NALU_TYPE_EOSTREAM`
13) `IH264_NALU_TYPE_FILLER`

This version defines one more NALU which is modified version of SPS nal unit, It is SPS having VUI

14) `IH264_NALU_TYPE_SPS_WITH_VUI`

B.3 Control masks

Encoder defines a control mask for each position in video sequence. Hence, it has following masks as part of `IH264ENC_NALUControlParams`, which can be configured as creation time

1) `naluPresentMaskStartOfSequence`
2) `naluPresentMaskIDRPicture`

3) naluPresentMaskIntraPicture

4) naluPresentMaskNonIntraPicture

5) naluPresentMaskEndOfSequence

Each of the mask is 14-bit mask with following bit allocation

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USER_DATA_UNREGD_SEI	SPS_VUI	FILLER	EOSTREAM	EOSEQ	AUD	PPS	SPS	SEI	IDR_SLICE	SLICE_DP_C	SLICE_DP_B	SLICE_DP_A	SLICE	UNSPECIFIED

Bit-0,1,2,3,4,5 are ignored in each mask

Since IDR picture is also an Intra picture, for IDR picture the nalUnitMask used by encoder is Oring of naluPresentMaskIDRPicture and naluPresentMaskIntraPicture.

Similarly, naluPresentMaskStartOfSequence also considers the properties of IDR picture

SEI (bit 6): This bit control the insertion of following SEI messages in video sequence. For details of these SEI messages refer Appendix D of H.264 standard

- timing_info_sei
- buffering_period_sei: This SEI is put only at IDR frames even if it is enabled for other positions in video sequence
- stereo_video_info_sei :This SEI is put only when stereoInfoPreset is enabled

If you want to encode with rateControlPreset == IVIDEO_NONE then nal_hrd_parameters_present_flag and vcl_hrd_parameters_present_flag will be false. Hence, buffering period SEI message will not be present.

SPS (bit 7): This bit controls the insertion of SPS in the video sequence. For a start of sequence SPS is must so encoder internally assumes this bit as 1 for naluPresentMaskStartOfSequence

PPS (bit 8): This bit controls the insertion of PPS in the video sequence. For a start of sequence, PPS is a must, hence, the encoder internally assumes this bit as 1 for naluPresentMaskStartOfSequence

AUD (bit 9): This bit controls the insertion of access unit delimiter NAL unit

EOSEQ (bit 10): This bit controls the insertion of end of sequence NAL unit. This bit is ignored for all the NAL unit masks except naluPresentMaskEndOfSequence.

EOSTREAM (bit 11): This bit controls the insertion of end of stream NAL unit. This bit is ignored for all the NAL unit masks except naluPresentMaskEndOfSequence.

FILLER(bit 12): This bit informs encoder to insert filler data. It is encoder's decision to put filler data or not based upon the constant bit rate need

SPS_VUI(bit 13): This bit informs encoder to insert SPS data along with VUI (Video usability Information).

USER_DATA_UNREGD_SEI (bit 14): This bit controls the insertion of user data unregistered SEI. To insert SEI some additional information has to be provided by user, refer Appendix A for more details

Bit-13 supersedes bit-3

B.4 End of Sequence Identification

Encoders don't know in general that this is the end of sequence position in video. Hence encoder except user to put it into flush mode to identify end of sequence.

When encoder is in flush mode it stops excepting input via process call and processes the buffers which it internally have (In case of B frame there are delays in producing the output hence encoder has some buffers unprocessed). Being in flush mode encoder knows that all the input buffers are exhausted or not and hence can decide the end of sequence

So when there is only P frames (no B frames) and still user want encoder to use `naluPresentMaskEndOfSequence`, he/she should call a control method with `XDM_FLUSH`

B.5 Erroneous Situations

Following are the situations, which are erroneous, for each of the situation encoder returns `IH264ENC_UNSUPPORTED_NALUNITCONTROLPARAMS` error code

1) If user want to encode the SEI, it is necessary to have SPS with VUI. So if user has configured SEI bit as 1 for some position in video sequence and there is no naluMask prior to that position having SPS_VUI enabled then encoder returns error

   For example:

```
naluPresentMaskStartOfSequence (bit-13 is 0, bit-6 is 1): This is
erroneous situation
```

```
naluPresentMaskStartOfSequence (bit-13 is 1, bit-6 is 0) and
naluPresentMaskNonIntraPicture (bit-13 is 0, bit-6 is 1): This is not
erroneous situation
```

2) If Bit-13 (SPS + VUI bit) in naluPresentMaskStartOfSequence is 0 then it should be 0 in all the remaining mask

3) If Bit-13 (SPS + VUI bit) in naluPresentMaskStartOfSequence is 1 then it should be 1 in all the mask which contains Bit-7 (SPS bit) as 1

4) If stereoInfoPreset is enabled (Stereo Video Coding) then inputcontenttype should be Interlaced.

Appendix C

Control for User Defined Scaling Matrices

This appendix explains the mechanism of supporting user defined scaling matrices.

Following operations are performed at different stages:

1) Creation time
2) Control time
3) Process level

C.1 Creation Time

The following parameters should be set during creation of encoder

1) `IVIDENC2_Params::metadataType[IDX_SCALINGMTX_METADATA] = IH264_USER_DEFINED_SCALINGMATRIX, here IDX_SCALINGMTX_METADATA can be any value between 0 to IVIDEO_MAX_NUM_METADATA_PLANES - 1.`

2) `IH264ENC_Params::IH264ENC_RateControlParams:: scalingMatrixPreset to be set as IH264_SCALINGMATRIX_USERDEFINED_SPSLEVEL or IH264_SCALINGMATRIX_USERDEFINED_PPSLEVEL`

`IH264_SCALINGMATRIX_USERDEFINED_SPSLEVEL` means that encoder will generate scaling matrices in bit-stream for each SPS

`IH264_SCALINGMATRIX_USERDEFINED_PPSLEVEL` means that encoder will generate scaling matrices in bit-stream for each PPS

```
typedef enum

{

 IH264 SCALINGMATRIX NONE        = 0 ,                     //!<
Flat Scaling matrix : part of standard  (NO Scaling Matrix)

 IH264 SCALINGMATRIX NORMAL      = 1 ,                     //!<
For normal contents

 IH264 SCALINGMATRIX DEFAULT     = IH264 SCALINGMATRIX NORMAL,
//!< default = IH264 SCALINGMATRIX NORMAL (if profile == HIGH) &
IH264 SCALINGMATRIX NONE (if profile!=HIGH).

 IH264 SCALINGMATRIX NOISY       = 2 ,                     //!<
For noisy contents

 IH264 SCALINGMATRIX STD DEFAULT = 3 ,                     //!<
Default Scaling Matrix provided by H.264 standard

 IH264 SCALINGMATRIX USERDEFINED SPSLEVEL = 4 , //!< User
defined SM at SPS level

 IH264 SCALINGMATRIX USERDEFINED PPSLEVEL = 5 , //! User defined
SM at PPS level
```

```
}
```

C.2 Control Time

Call Control function with `XDM_GETBUFINFO`. Encoder should return one additional input buffer as required. Size of the buffer will be 896 bytes

C.3 Process level

Application should have memory allocated for this meta data and pass on to the encoder

```
via IVIDEO2 BufDesc *inBufs->numMetaPlanes = 1

inBufs->metadataPlaneDesc[IDX SCALINGMTX METADATA].buf =
pBuffer ;
inBufs->metadataPlaneDesc[index].bufSize.bytes = 896
```

If application want to provide the size as part of meta data then it should set `inBufs->metadataPlaneDesc[index].bufSize.bytes = -1` otherwise encoder will read the size from `metadataPlaneDesc[index].bufSize.bytes` field.

Index of `metadataPlaneDesc` follows these rules:

Note:

- Encoder assumes the availability of payload during processing of entire sequence, if it is user defined.
- Encoder assumes that for each process call the payload is provided.

C.3.1 *Format of Payload*

```
typedef struct

{

 U16 wgt4x4[2][3][2][4][4];
//[Intra(0)/Inter(1)][Y(0)/Cb(1)/Cr(2)][Inv(0)/Fwd(1)][4][4]

U16 wgt8x8[2][2][8][8];
//[intra(0)/inter(1)][inv(0)/Fwd(1)][8][8]

} sH264WgtTables t ;
```

Comments in above structure explain the usage of each dimension. For example, forward Intra Chroma Cr component is pointed by `sH264WgtTables_t::wgt4x4[0][2][1][4][4];`

Inv/Fwd are explained below:

- **Inv:** This means the actual scaling matrices which decoder derives after decoding from the bit-stream
- **Fwd:** This is a derived value from Inv data which is used by encoder in Forward path, it is 1/Inv value in Q.18 format and only lower 16 bits are considered (upper two bits are always 0)

Example:

```
Inv =
```

```
{{16, 16, 16, 16}, {16, 16, 16, 16},{16, 16, 16, 16},{16, 16,
16, 16}};

Fwd =

{

{

MIN((0x40000 + 16/2)/16, 0xFFFF), MIN((0x40000 + 16/2)/16,
0xFFFF), MIN((0x40000 + 16/2)/16, 0xFFFF), MIN((0x40000 +
16/2)/16, 0xFFFF)

},

{

MIN((0x40000 + 16/2)/16, 0xFFFF), MIN((0x40000 + 16/2)/16,
0xFFFF), MIN((0x40000 + 16/2)/16, 0xFFFF), MIN((0x40000 +
16/2)/16, 0xFFFF)

},

{

MIN((0x40000 + 16/2)/16, 0xFFFF), MIN((0x40000 + 16/2)/16,
0xFFFF), MIN((0x40000 + 16/2)/16, 0xFFFF), MIN((0x40000 +
16/2)/16, 0xFFFF)

},

{

MIN((0x40000 + 16/2)/16, 0xFFFF), MIN((0x40000 + 16/2)/16,
0xFFFF), MIN((0x40000 + 16/2)/16, 0xFFFF), MIN((0x40000 +
16/2)/16, 0xFFFF)

},

};
```

C.3.2 Constraints on Payload Data

Each value has to be an unsigned 16-bit value. As per formula to compute forward matrix value, the minimum value for scaling matrix weight in inverse path is 4.

Maximum value for scaling matrix weight in inverse path is 255

There is no error check performed for the values of scaling matrices and the behavior is not defined for non-supported values.

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 non-zero value indicates that the analytic info is needed. When this parameter is set to non-zero value, 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 10 bytes, a signed horizontal displacement component (signed 16-bit integer) and a vertical displacement component (signed 16-bit integer) in L0 and L1 direction and SAD (16-bit integer).

The following sequence should be followed for Analytic Info access:

17) In the create time parameters, set the flag to access analytic data

```
/* Enable MV access */
createParams ->enableAnalyticinfo = 1;
```

18) Allocate output buffers and define the output buffer descriptors

```
/* Output Buffer Descriptor variables */
XDM2_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.descs[0].buf = streamDataPtr; //pointer to
H264 bit-stream
outputBufDesc.descs[0].bufSize.bytes =
status.videnc2Status.bufInfo.minOutBufSize[0].bytes;


/* MV & SAD Buffer */
outputBufDesc.descs[1].buf = Output Buffer Base Addr;
//pointer to MV and SAD data
```

```
outputBufDesc.descs[1].bufSize.bytes =
status.videnc2Status.bufInfo.minOutBufSize[1].bytes;
```

19) Call frame encode API

```
/* Process call to encode 1 frame */
process(..  ,..  , outputBufDesc, .. );
```

After this call, the buffer `outputBufDesc.descs[1].buf` will have SAD and Motion vector data. The data format of this buffer will be like,

AnalyticHeaderInfo	Data (SAD and MV)

Define a structure:

```
struct AnalyticHeaderInfo
{
   U32 NumElements;
   ElementInfo elementInfoField0SAD;
   ElementInfo elementInfoField1SAD;
   ElementInfo elementInfoField0MVL0;
   ElementInfo elementInfoField0MVL1;
   ElementInfo elementInfoField1MVL0;
   ElementInfo elementInfoField1MVL1;
} ;

Where as
NumElements -> Total number of elements in the buffer
(As of now SAD ,MV in L0 direction and MV in L1
direction for each field in case of interlace content)

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 */
  U32 Jump;
    /* Number of data elements in this group */
  U32 Count;
}ElementInfo;
```

The data format will differ for each frame type; there can be four different formats as,

1. Process call which generates one P frame/field
2. Process call which generates two P fields
3. Process call which generates one B frame/field
4. Process call which generates two B fields

Note: The data present in the **shaded boxes** of the below figures are **don't care** values.

Process call, which generates one P frame/field:

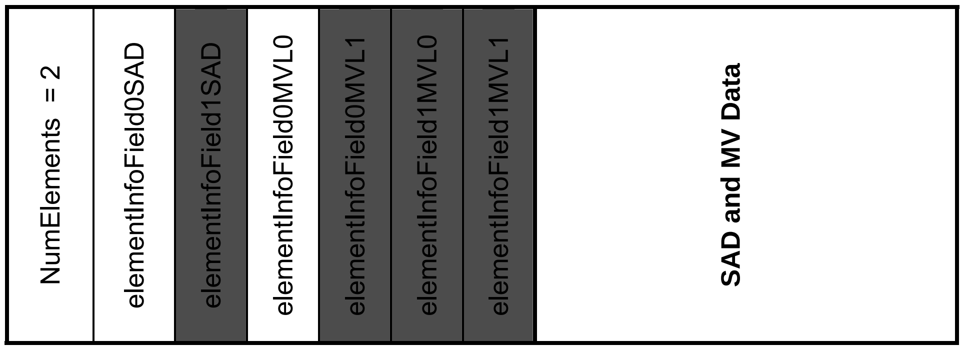


Figure D-1. Data format of Analytic Information in case of P frame/field.

Process call, which generates two P fields:

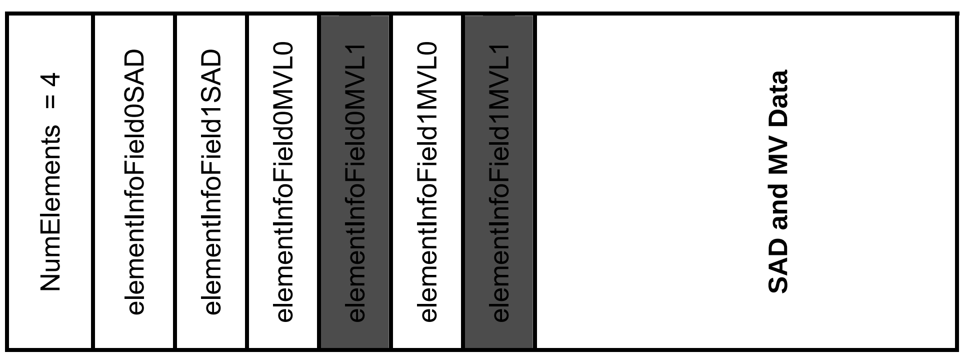


Figure D-2. Data format of Analytic Information in case of two P fields.

Process call, which generates one B frame/field:

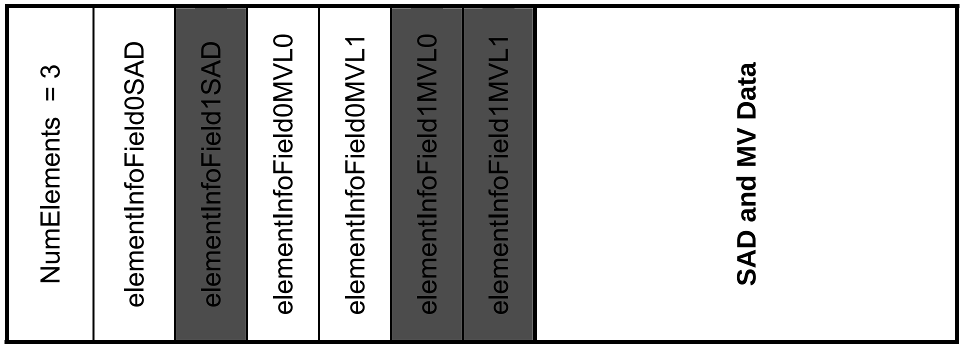


Figure D-3. Data format of Analytic Information in case of B frame/field

Process call, which generates two B fields:

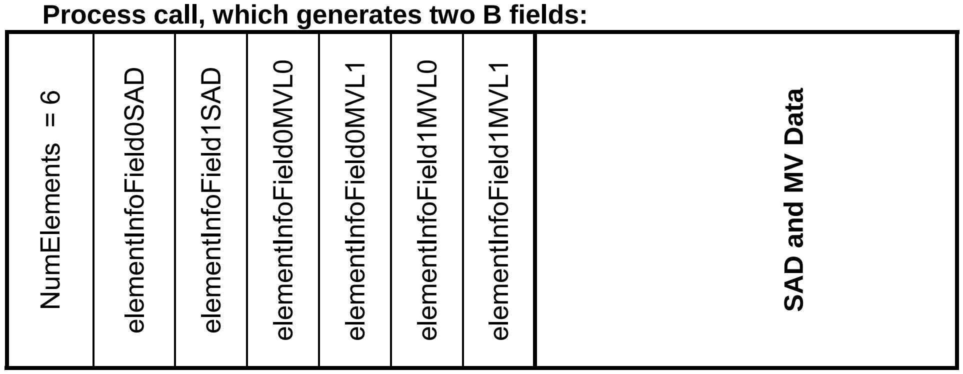


Figure D-4. Data format of Analytic Information in case of two B fields.

D.2 Example Usage

For example, data in output buffer dumped by the codec for a progressive - B frame is as shown below,

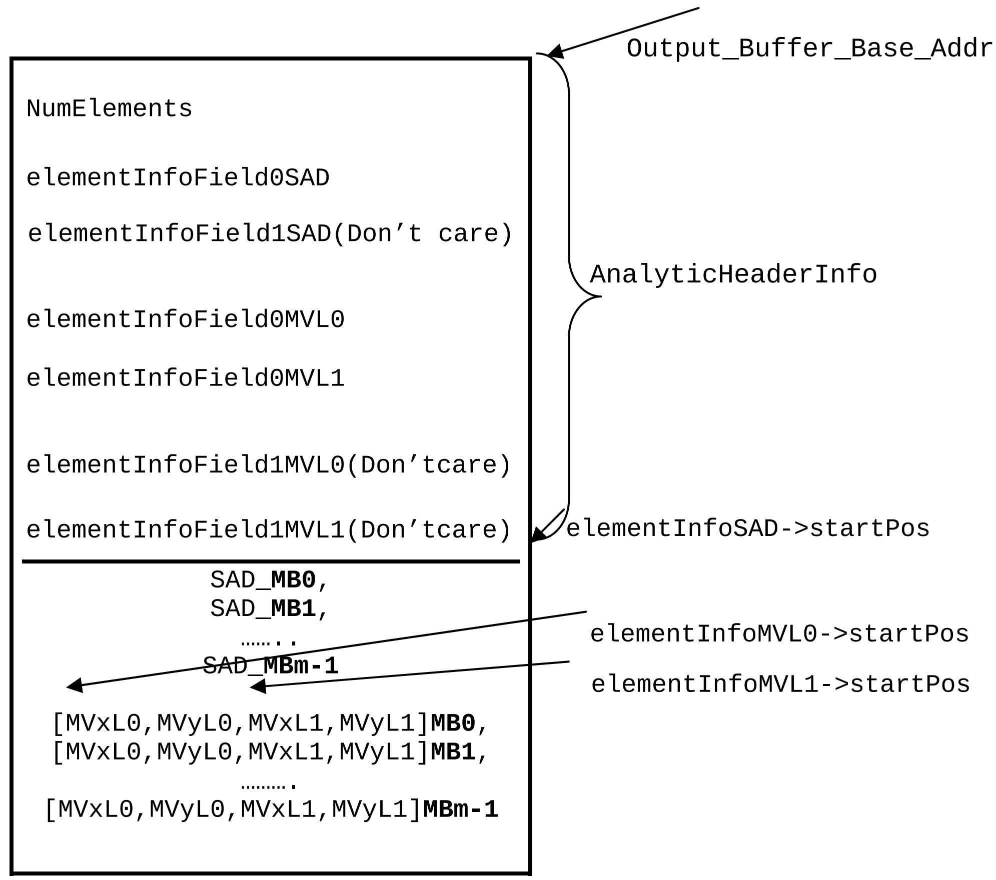


Figure D-5. MV and SAD data dump by codec in case of progressive B frames.

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

```
S16 *Src = (U32)Output_Buffer_Base_Addr +
    elementInfoMVL0->StartPos;
U32 Jump = elementInfoMVL0->Jump;
S16 *MVL0 = Addr_to_store_MV_inL0 ;
Jump = Jump / sizeof(S16);

for (i = 0; i < elementInfoMVL0->Count; i = i++)
{
   * MVL0 ++ = Src[i * Jump]; // To get MVx
   * MVL0 ++ = Src[((i *Jump) + 1)]; //To get MVy
}
```

Note:

- The motion vectors are with quaterpel resolution.
- 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 decission. The MV buffer returns the results of the motion estimation in quaterpel resolution (lowest SAD) and this may be different from the motion vectors seen in the bit-stream. More details are given below :
- Some 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.
- Due to the post motion estimation decisions for some macro blocks, the actual motion vector encoded may be forced to skip MV. In this case, the non-skip motion vector available after the motion estimation is returned.
- For I-frames, motion vectors and SAD are not present in the buffer.

Appendix E

Debug Trace Support

This appendix explains the Debug Trace support details on encoder. This is to help the application to get the trace data generated by Encoder from external memory

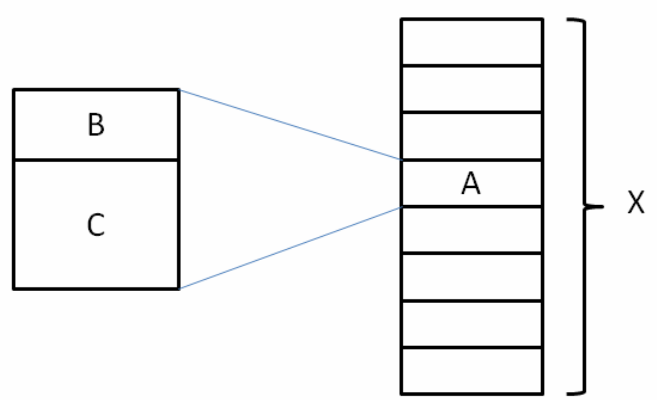
E.1 Debug Trace design in Encoder

Encoder has "`debugTraceLevel`" interface to select the debug trace level. When "`debugTraceLevel`" is set to zero then Encoder will not generate any trace data. Otherwise, it will generate the trace data in external memory. Encoder has support to log last N frame's debug trace data which is controlled by "`lastNFramesToLog`" interface parameter.

If the encoder is requested to generate debug trace data then encoder will request for external memory to store these trace data. The size of this memory depends on the "`lastNFramesToLog`" parameter. If the size for one process calls trace data is A bytes then the total bytes requested will be

Total size (X) = (1 + `lastNFramesToLog`) * A bytes.

Each instance of this trace buffer has two sections of trace data. First section (B) is written in external memory by Media Controller through cache and other section (C) is written by HDVICP2.0 using DMA.



Here size of both B and C are aligned to cache line size, which is 32 bytes. Codec will not do any cache related operation at any point of time. Since the section B is written by Media Controller through cache, cache write back needs to be performed for this section. If application has programmed the `lastNFramesToLog` values as N, then the cache write back needs to be performed N+1 times. HDVICP2.0 will write section C using DMA, so the cache write back is not required for this section.

E.1.1 Steps to utilize debug trace support in H264 encoder

Create encoder with following settings

```
IH264ENC_Params.debugTraceLevel     = 1;
IH264ENC_Params.lastNFramesToLog    = N; (example: 10)
```

Then make a control call with "XDM_GETSTATUS" command to get the following parameters from codec

Debug trace level used by codec

```
IH264ENC_Status. debugTraceLevel
```

Number of frames for which log is available

```
IH264ENC_Status. lastNFramesToLog
```

Base address of trace data in external memory

```
IH264ENC_Status.extMemoryDebugTraceAddr
```

Total size of trace buffer in external memory

```
IH264ENC_Status.extMemoryDebugTraceSize
```

Size of trace buffer for one process call (A) is

```
A = IH264ENC_Status.extMemoryDebugTraceSize / (IH264ENC_Status.
lastNFramesToLog + 1)
```

Cache write back operation needs to be performed before reading this data from external memory.

Pseudo code for cache write back

```
ddrAddress = IH264ENC_Status.extMemoryDebugTraceAddr;

totalNumFrames = (IH264ENC_Status.lastNFramesToLog + 1);

for(i = 0; i < totalNumFrames i++)

{

      CacheWriteBack(ddrAddress,B);

      ddrAddress += A;

}
```

Definition of "CacheWriteBack" function

```
CacheWriteBack(void * address, int size);
```

Here

`address` : Start address for write back operation

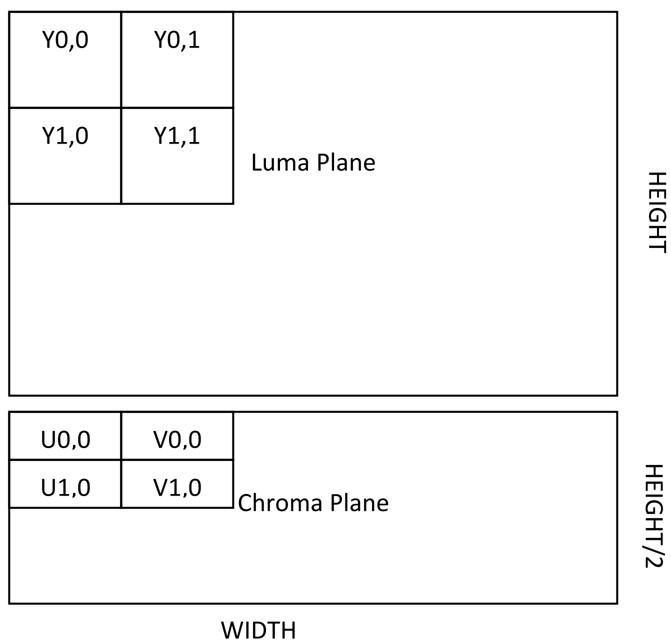
`Size` : length in bytes

Picture format

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

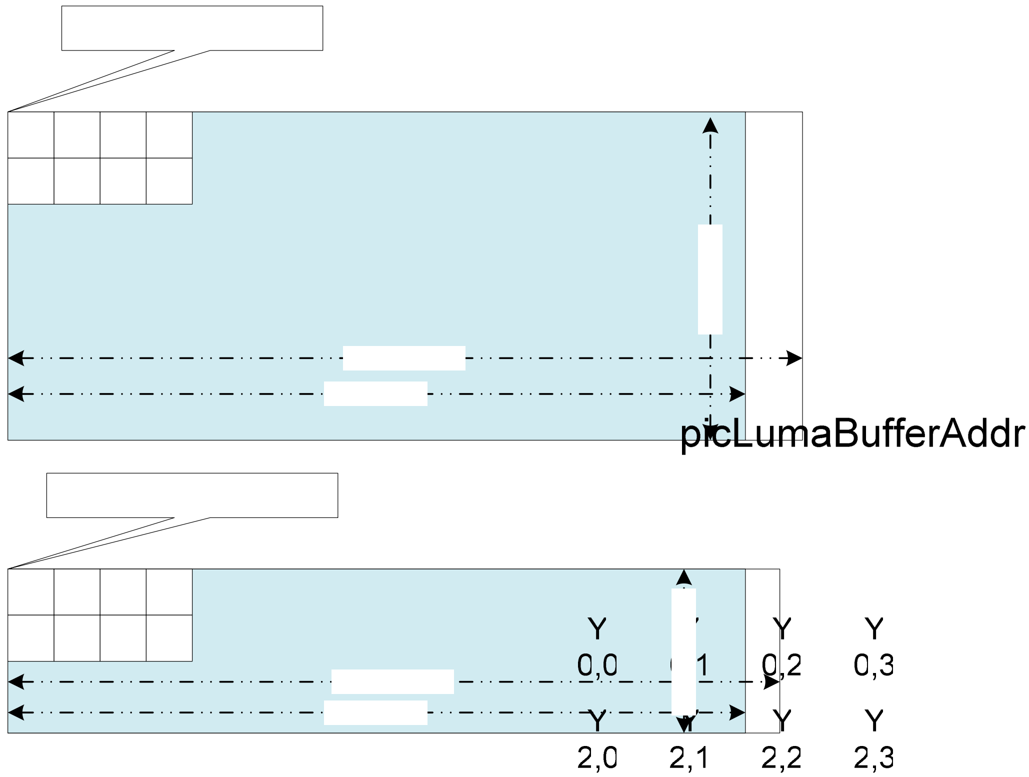
F.1 NV12 Chroma Format

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



F.2 Progressive and Interlaced Format

F.2.1 Progressive Format

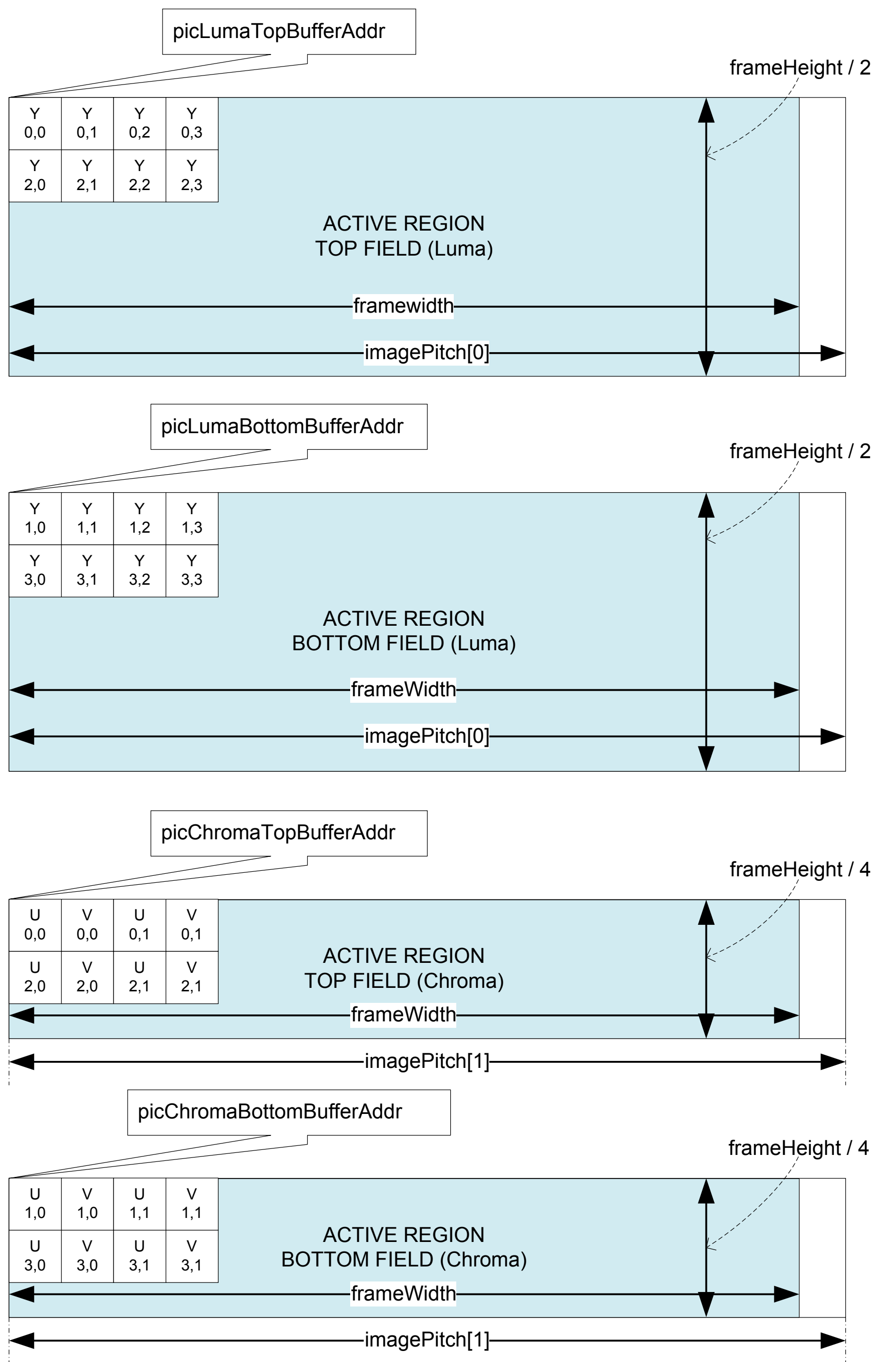


ActiveRegion: Data to be encoded

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

F.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, Encoder also supports filed interleaved data format where both filed are interleaved in memory.

F.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 = 4352
- Maximum frameHeight = 4096
- frameWidth shall be a multiple of 16 bytes
- frameHeight shall be multiple of 2

Interlaced:

- Minimum frameWidth = 96
- Minimum (frameHeight/2) = 80
- Maximum frameWidth = 4352
- Maximum (frameHeight/2) = 4096
- frameWidth shall be a multiple of 16 bytes
- frameHeight shall be multiple of 4.

Appendix G

Low Latency / Sub Frame Level Synchronization

This appendix explains the details of H264 encoder’s low latency features and how to exercise them.

G.1 Description

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

This appendix explains the sub-frame level data communication capabilities of video codec using data synch call backs defined with IVIDENC2 interface

G.2 H.264 Encoder Input with sub frame level synchronization

H.264 encoder allows accepting partial frames for the application on input side and can start encoding. This section explains the IVIDENC2 interface details which help to achieve the sub frame level communications on input side of a video encoder.

Table 20 , Table 21 and Table 22 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 20 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
IVIDENC2_Params::numInputDataUnits	Unit of input data	Don’t care. As the information about the data can be available during sub frame level communication	

Table 21 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 22 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 directly to the start of the first block, and is not treated as a pointer to an array. If scatteredBlocksFlag is set to XDAS_TRUE, this field points to an array of pointers to the data blocks.	**Don't care** since it is assumed to be contigous yuv buffer and initial address is via inbuf at process call.

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

Constraint

In presence of B frame, IVIDENC2_Params::inputDataMode = IVIDEO_NUMROWS is an erroneous case

IVIDENC2_DynamicParams::getDataFxn == NULL && IVIDENC2_Params::inputDataMode == IVIDEO_NUMROWS is an erroneous situation and codec returns error during process call.

G.3 H.264 Encoder Output with sub frame level synchronization

H.264 encoder allows providing partial compressed bit-stream to the application on output side. This section explains the IVIDENC2 interface details, which help to achieve the sub frame level communications on output side of a video encoder.

Table 23, Table 24 explain the creation and control 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 23 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 1K
		IVIDEO_SLICEMODE	bit-stream is provided by encoder after producing a single(or more) number of NAL Units
IVIDENC2_Params::numOutputDataUnits	Unit of output data	**Don't care** if outputDataMode == IVIDEO_ENTIREFRAME	
		Any positive value if outputDataMode != IVIDEO_ENTIREFRAME if outputDataMode == IVIDEO_FIXEDLENGTH then it indicates the basic unit of size (in multiple of 1K) at which encoder should inform the application. Eg: Here 4 means that encoder should inform after producing every 4*1024 bytes to application if outputDataMode == IVIDEO_SLICEMODE then it indicates the basic unit of slices at which encoder should produce the bit-stream. Eg: Here 5 means that after encoding a set of 5 NALUs, encoder should inform to application	

Table 24 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 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.	Any Value

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

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

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

It should also make a control call with IVIDENC2_DynamicParams::putDataFxn = non-NULL; to use sub frame level data communication, control call is mandatory.

It should provide the base address and available space of the output buffer during process call

Erroneous case

IVIDENC2_DynamicParams::putDataFxn == NULL && IVIDENC2_Params::outputDataMode != IVIDEO_ENTIREFRAME is an erroneous situation and codec returns error during process call.

If outPutDataMode == IVIDEO_SLICE and multiple slices are not enabled (sliceMode == IH264_SLICEMODE_NONE), encoder returns error (IH264ENC_UNSUPPORTED_SLICECODINGPARAMS) during create time

If numOutputDataUnits > 64 or numOutputDataUnits < 0 with outputDataMode != IVIDEO_ENTIREFRAME is an erroneous situation and code returns error (IH264ENC_IMPROPER_DATASYNC_SETTING) during create time

If Number of B frame > 0 && inputDataMode != IVIDEO_ENTIREFRAME, encoder returns error (IH264ENC_IMPROPER_DATASYNC_SETTING) at create time

If minBitRate > 0 && outputDataMode != IVIDEO_ENTIREFRAME, encoder returns error (IH264ENC_UNSUPPORTED_VIDENC2PARAMS) at create time

If outPutDataMode == IVIDEO_SLICE   and sliceMode = IH264_SLICEMODE_BYTES, then encoder expects getBufferFxn to be implemeneted by application. So outPutDataMode == IVIDEO_SLICE   && sliceMode = IH264_SLICEMODE_BYTES && getBufferFxn == NULL) is erroneous condition and encoder returns IH264ENC_IMPROPER_DATASYNC_SETTING error during control call

G.3.1 H.264 Encoder mechansim to accpet partial buffer and non contiguous buffer on output side

Before unserstanding, the interface related to different outputDataMode, it is important to understand about the interface, which allows encoder to accept non-contiguous memory

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 25 and Table 26 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.

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

getBuf is independent of outputDataMode or inputDataMode. It is only meant for codec to ask application for a buffer, if encoder has exhausted for output bit-stream

During process call the initial stream address and size are provided by application. No constraint on this information and encoder consumes this buffer space

During data synch (via getBuf) codec can accept a multiple non contiguous buffers from application each of them has to be multiple of 2K. (only exception here is when encoder is congiured to work with outputDataMode = IVIDEO_SLICE_MODE and outputDataMode == IVIDEO_SLICE_MODE. With this case encoder can accept any size which is >= sliceUnitSize)

if scatteredBlocksFlag is non zero then 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 (**constraint**)

If scatteredBlocksFlag flag is zero than there is no limit on numBlocks.

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).

Table 25 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 datasync should set this to NULL. For an algorithm, this handle is read-only; it must not be modified when calling the app-registered IVIDENC2_DynamicParams.getBufferFxn(). The app/framework can use this handle to differentiate callbacks from different algorithms.	Any Value

Table 26 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 if scatteredBlocksFlag != 0 then values higher than 8 are assumed to be 8
XDM_DataSyncDesc::varBlockSizeFlag	Flag indicating whether any of the data blocks vary in size.	XDAS_TRUE or XDAS_FALSE
XDM_DataSyncDesc::blockSizes	Variable block sizes array.	non-NULL. If it is NULL then again call back definition of blockSize[i] is different for different situations as mentioned below - For sliceMode = IH264_SLICEMODE_BYTES and outputDataMode == IVIDEO_SLICE_MODE it should hold a value >= sliceUnitSize - For other situations it should hold a value which is multiple of 2K If application doesn't Obey these restrictions then the behavior is undefined

		totalBlockSize = SUM(blockSizes[0] to blockSizes[numBlocks-1]) if varBlockSizesFlag is non zero. totalBlockSize = numBlocks * blockSizes[0] if varBlockSizesFlag is zero if totalBlocksSize is 0 the call back again

G.3.2 H.264 Encoder behavior with outputDataMode as IVIDEO_SLICEMODE

Table 27 explains the handshake parameters for sub frame level data communication with outputDataMode = **IVIDEO_SLICEMODE**

Communication point by codec to application about data availability is one of the below whichever is **later**

numof slices(numOutputDataUnit) is encoded i.e. if in the current page ,numOfSlice >= numOutputDataUnit then make a putData call.

Minimum 1K of data is encoded i.e. numOfSlices exceeds numOutputDataUnit in the first page cross itself.

Note that communication point is always at page cross over except at the last call (end of process) where bit stream can end at any point in the page.

Incase of **outputDataMode = IVIDEO_SLICEMODE**, following points should be noted

numOutputDataUnit is the frequency after which codec will inform to App. So in IVIDEO_SLICE_MODE, lets outputDataUnit is 8 then after 8 slice codec has to make putData call.

This encoder implementations has **constraint** of limiting maximum allowed value of outputDataUnit as 64

Let's say numOutputDataUnit is 64, and in one page codec generates 63 slices and in the next page it generated again 64 slices, in this case codec will inform all the 127 slices. So maximum generated value by encoder for numBlocks is 127

Bit-stream can be non-contiguous at NAL boundaries, if the encoder is configured to generate NAL Units of fixed length (sliceMode == IH264_SLICEMODE_BYTES). In this case after each NALU completion, encoder moves to next NALU's start address even there are few bytes left in the previous buffer(packet)

If the encoder is configured to generate slices based upon macroBlockPerSlice (sliceMode ==IH264_SLICEMODE_MBS or sliceMode == IH264_SLICEMODE_OFFSET) then the bit-stream is assumed to be contiguous in memory, hence it is user's responsibility to provide the bit-stream address during data synch calls(XDM_DataSyncDesc::baseAddr) to be in continuation of the earlier bit-stream address provided to encoder

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)

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 27 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: None
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.	This field points directly to the start of the data for the active transaction
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. Codec can generate following possible values of numBblocks 1 <= numBlocks <= 127
XDM_DataSyncDesc::varBlockSizeFlag	Flag indicating whether any of the data blocks vary in size.	XDAS_TRUE or XDAS_FALSE(slice sizes are not constant most of the time)
XDM_DataSyncDesc::blockSizes	Variable block sizes array.	If varBlockSizesFlag is XDAS_TRUE, this array contains the sizes of each slice. So total 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])

G.3.3 H.264 Encoder behavior with outputDataMode as IVIDEO_FIXEDLENGTH

Table 28 explains the handshake parameters for sub frame level data communication with outputDataMode = **IVIDEO_FIXEDLENGTH**

Communication point by codec to application about data availability is one of the below whichever is **earlier**

1 K Bytes * numOutputDataUnit of data is encoded.

if 64 non-continuous blocks have been generated by encoder.

Note that communication point is always at page cross over except at the last call (end of process) where bit stream can end at any point in the page.

Incase of **outputDataMode = IVIDEO_FIXEDLENGTH**, following points should be noted

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

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

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

Table 28 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 block may be scattered in memory. XDAS_TRUE or XDAS_FALSE
XDM_DataSyncDesc::baseAddr	Base address of single data block or pointer to an array of data block addresses of size numBlocks. If scatteredBlocksFlag is set to XDAS_FALSE, this field points directly to the start of the first block, and is not treated as a pointer to an array. If scatteredBlocksFlag is set to XDAS_TRUE, this field points to an array of pointers to the data blocks.	Base address of single data block or pointer to an array of block addresses of size numBlocks. If scatteredBlocksFlag is set to XDAS_FALSE, this field points directly to the start of the first block, and is not treated as a pointer to an array. If scatteredBlocksFlag is set to XDAS_TRUE, this field

		points to an array of pointers to the data blocks i.e. from baseAddr[0] to baseAddr[numBlocks-1]
XDM_DataSyncDesc::numBlocks	Number of data blocks	It is the number of blocks generated till the point of putData call. Codec can generate following possible values of numBblocks 1 <= numBlocks <= 64
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])

This page is intentionally left blank

Appendix H

Long Term Reference Picture Schemes

This appendix explains the usage details of long-term reference picture support.

H.1 Description

Most of the applications which are sensitive to errors over network need error resilient features in the vide encoder. Long-term reference picture allows encoder to prevent propogation of erros in few temporal frames in past. Most of the multi-way real time communication systems can get benifited with this error resiliency feature in video encoders.

H.2 Supported Schemes and Usage

This version of encoder supports following mechanisms for long-term reference picture,

1. Periodic Long term Reference picture
2. Proactive Long term Referencing
3. Reactive Long term Referencing

H.2.1 Periodic Long term Referencing

This scheme allows encoder to get instructed to refer to last marked long term reference picture. Pictures are marked as long term reference picture based on the given period. To enable this scheme following operations should be performed at create time IH264ENC_Params::enableLongTermRefFrame should be set to IH264ENC_LTRP_REFERTO_PERIODICLTRP.

IH264ENC_Params::LTRPPeriod should be set to long term reference picture marking interval i.e, interval between two consecutive long term reference pictures.

This will cause encoder instance memory to be higher by one frame than in normal operation. If normal operation is requiring 2 reference frames, now it will be require 3 reference frames to be stored.

At process level IVIDENC2_InArgs::control should be set to IH264ENC_CTRL_REFER_LONG_TERM_FRAME for the desired frame to refer to lastly marked long term reference picture, so user has flexibility for each encoding frame to inform to encoder to use lastly marked long term reference picture. The below picture explains how long term reference pictures are marked based on LTRP Period and IntraframeInterval.

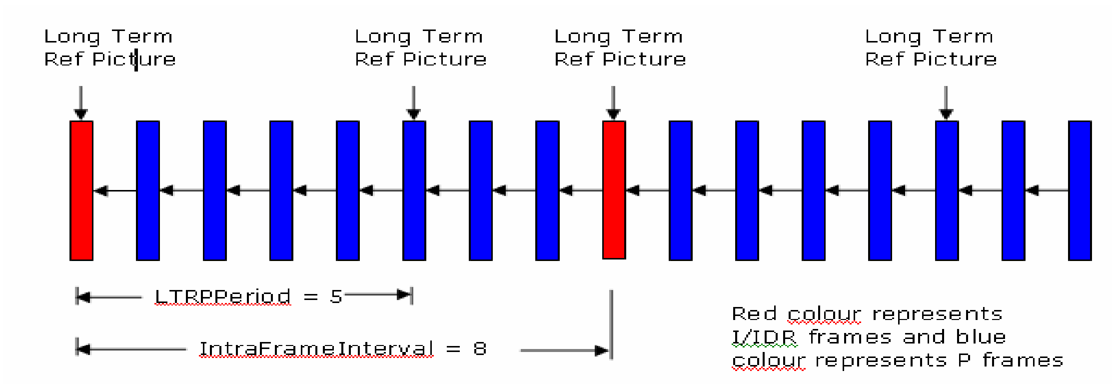


Figure H-1. Marking of Long Term reference picture

Next two figures explains the usages of this feature in a 2 way video transmission system

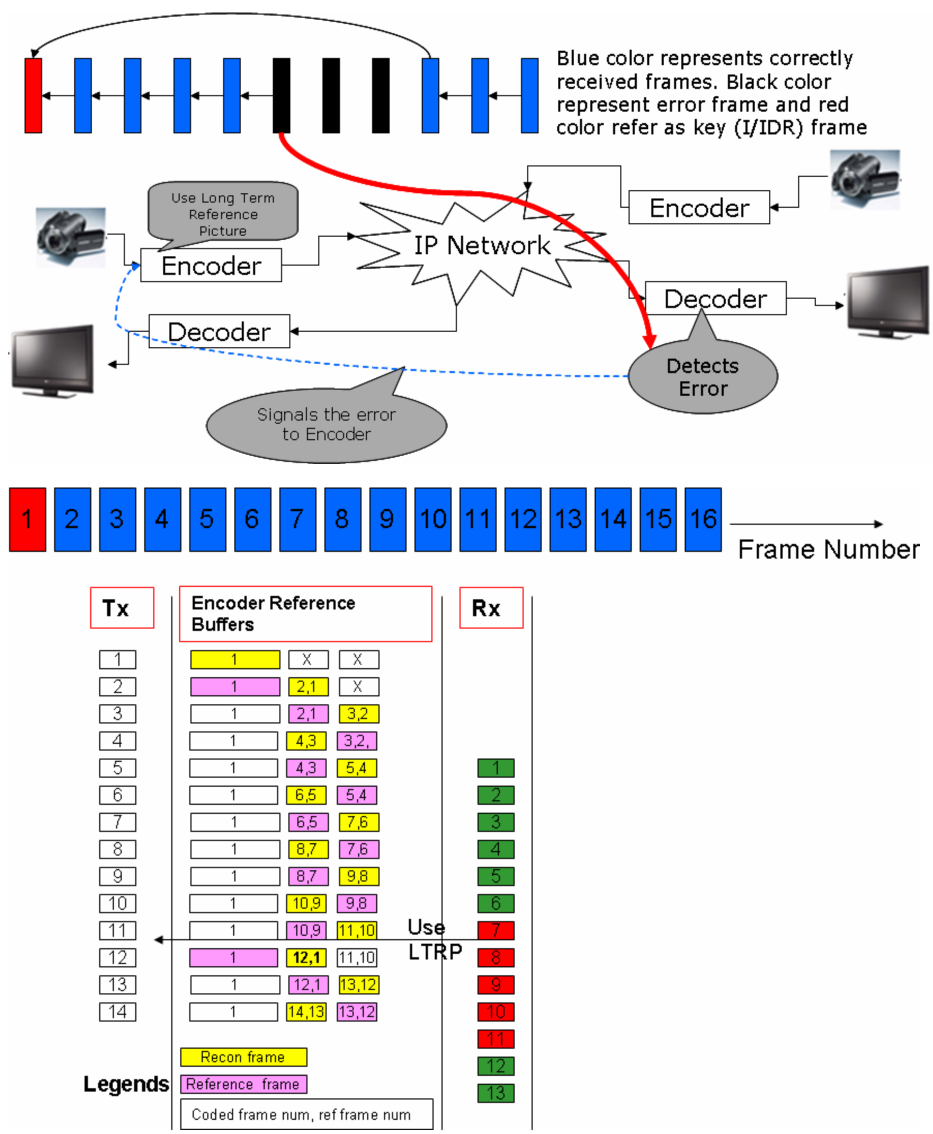


Figure H-2. Long term referencing to I/IDR - 2 way video transmission system.

H.2.2 Proactive Long term Referencing

This scheme allows encoder to be instructed to refer to create a structure with which it is reactive to errors. In previous scheme, an action is taken after the error gets introduced and recognized from the receiver, where as in this scheme it allows to create a bit-stream gop structure which is recoverable in presense of errors.

To enable this scheme following operations should be performed:

At create time IH264ENC_Params::enableLongTermRefFrame should be set to IH264ENC_LTRP_REFERTOP_PROACTIVE. This will not cause encoder instance memory to be higher than normal operation. For IPPP.. kind of sequence 2 frame buffers will be required as in normal scenarion. In normal scenarion when there is no long term referencing is enabled than one buffer is used to write the reconstructed data for current frame that is being encoded and another buffer is used as reference frame for current frame. In this kind of long term referencing scheme, among the two buffers only one buffer will be reference frame, another buffer will be kept for future usages. User can control which frame to be reconstructed and for which frame reference frame needs to be changed.

At process level IVIDENC2_InArgs::control should be set to either of the 4 values based upon the need

IH264ENC_CTRL_NOWRITE_NOREFUPDATE

IH264ENC_CTRL_WRITE_NOREFUPDATE

IH264ENC_CTRL_NOWRITE_REFUPDATE

IH264ENC_CTRL_WRITE_REFUPDATE

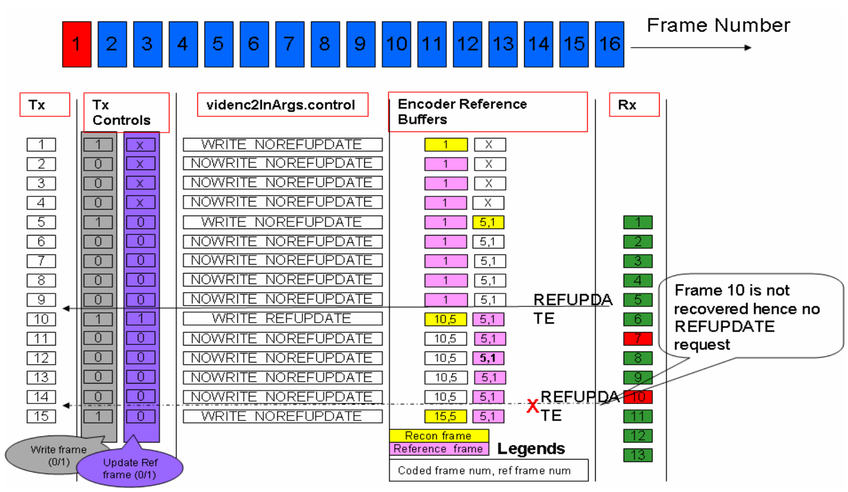
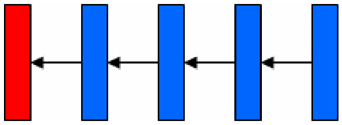


Figure H-3. Proactive Long term Referencing.

In the above figure, while encoding 5th frame user has given control to write the reconstructed frame, so that is why in this process call 5th frame got reconstructed and placed in one of the buffer, Non reference frame is always flushed if needed for storing new reconstructed frame. In the above example till the encoding of 9th frame, frame number ‘1’ was used as reference. In 10th frame encoding user has given the control about to update the refrence, so from this frame onward 5th frame will be used as reference.

One can acheieve different GOP structure with different values of IVIDENC2_InArgs::control at frame level

If one sets IVIDENC2_InArgs::control = IH264ENC_CTRL_WRITE_REFUPDATE then below GOP structure is achieved. This is equivalent to normal gop structure with no long term referencing



Based upon the feedback from receiver, one can achieve dynamically either one of the below gop structure.

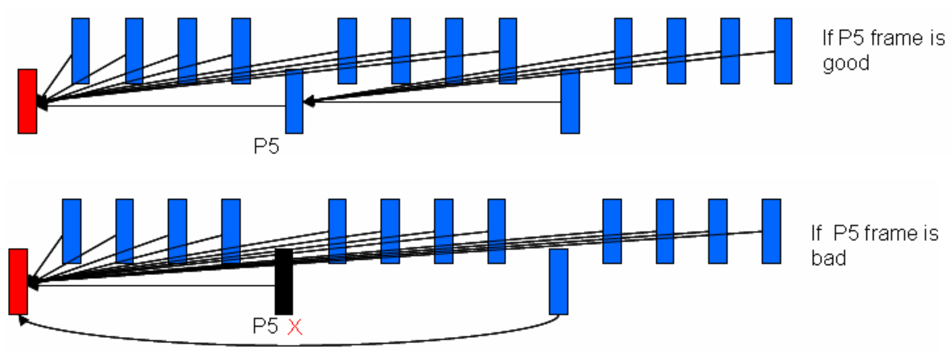
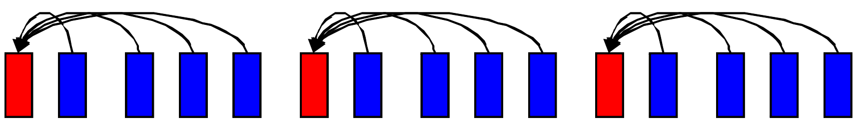


Figure H-4.GOP structure in LTRP.

With this control, user can achieve unchained P frame in which all P frame refers to only I frame. This GOP structure is useful for storage thinning over the time. In video security domain, if the content is aged it can be thnned by removing any of the P frames by video editing in below GOP structure.



Since the control is provided at picture level, it gives a lot of flexibility to application for getting desired gop structure dynamically. Below table provides the value of control field to achieve both situations.

	Frame #	If P5 is well received from receiver	If P5 is not well received from receiver
IVIDENC2_InArgs::control	0	IH264ENC_CTRL_WRITE_REFUPDATE	IH264ENC_CTRL_WRITE_REFUPDATE
	1	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE
	2	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE
	3	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE
	4	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE
	5	IH264ENC_CTRL_WRITE_NOREFUPDATE	IH264ENC_CTRL_WRITE_NOREFUPDATE
	6	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE
	7	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE
	8	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE
	9	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE
	10	IH264ENC_CTRL_WRITE_REFUPDATE	IH264ENC_CTRL_WRITE_NOREFUPDATE
	11	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE
	12	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE
	13	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE
	14	IH264ENC_CTRL_NOWRITE_NOREFUPDATE	IH264ENC_CTRL_NOWRITE_NOREFUPDATE

H.2.3 Reactive Long term Referencing

The operation of this scheme is similar to Long term Referencing to PeriodicLTRP scheme (H.2.1). This scheme differs by maintaining two recent long term reference pictures at any point of time. When encoder gets control command to refer to long term picture, it will refer to old long term reference picture from the recently marked 2 long term reference picture. Pictures are marked as long term reference picture based on the given period

To enable this scheme following operations should be performed at create time IH264ENC_Params::enableLongTermRefFrame should be set to IH264ENC_LTRP_REFERTOP_REACTIVE.

IH264ENC_Params::LTRPPeriod should be set to long term reference picture marking interval i.e, interval between two consecutive long term reference pictures.

This will cause encoder instance memory to be higher by two frames than in normal operation. If normal operation is requiring 2 reference frames, now it will be require 4 reference frames to be stored.

Next two figures explains the usages of this feature in a 2 way video transmission system

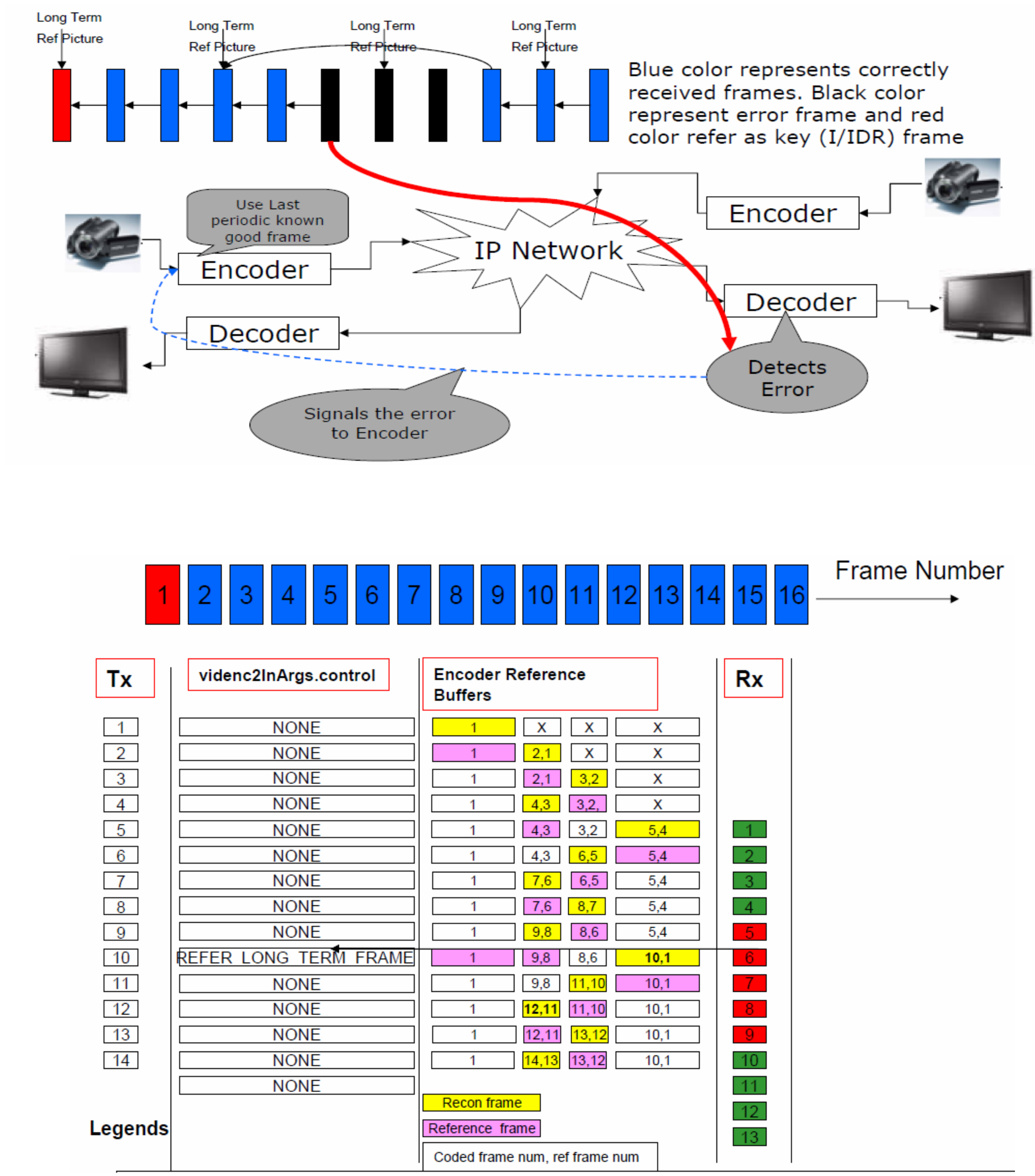


Figure H-5. Reactive Long term Referencing - 2 way video transmission system.

Appendix I

Hierarchical P structure Coding Scheme

This appendix explains the usage details of Hierarchical P structure coding.

I.1 Description

Hierarchical P structure allows additional flexibility to have a scalable bit-stream in terms of bit rate and frame rate without adding any additional delay. With this structure, pictures are coded in different temporal layers, where a picture only refers to pictures belonging to layers below it for temporal prediction

Below figure I.1 shows the temporalLayer 4 structure coding (i.e from temporal layer 0 to 3).

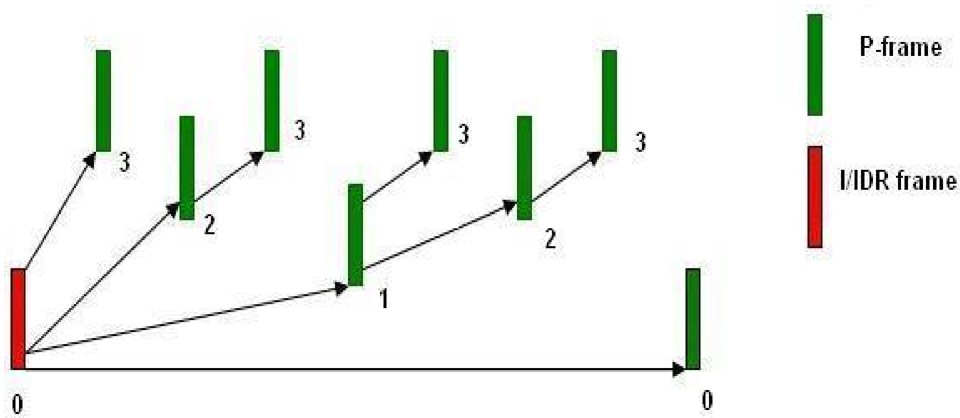


Figure I-1. Hierarchical P structure coding for Temporal Layer 4 with layer numbers

By removing the higher layer pictures, one can achieve a decodable bit-stream with lesser frame rate and hence lesser bitrate as well. In above example by removing the pictures from layer 3, the resultant bit-stream is of half the frame rate and almost half the bit-rate of original bit-stream

I.2 Supported Schemes and Usage

This version of encoder supports following mechansims for Hierarchical P structure coding.The fig I.1 Hierarchical P structural coding can be generated using two referencing schemes which are mentioned below.

I.2.1 Long term Referencing (MMCO Commands)

This scheme get enabled when `referencePicMarking == 1`.In this mode of operation,DPB management is done by long term frames. LongTerm pictures have longTermIndex correspoding to their layer.MMCO Commands get used for this operation.

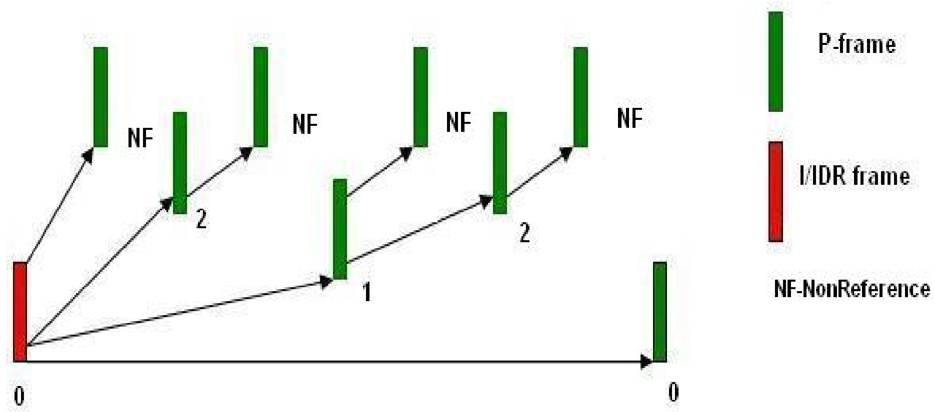


Figure I-2. Hierarchical P structure coding for Temporal Layer 4 with MMCO Commands

For interlaced coding, LT 0 and LT1 are toggled for base layer and LT2 and LT3 used for higher(Enhancement) layers. If `interlaceCodingType` is coded using MRF(Most recent field referencing scheme) then top layer will not be Non-Reference frame.

I.2.2 Short term Referencing (Sliding Window)

This scheme get enabled when `referencePicMarking == 0`. In this mode of operation,DPB management is done by short term frames (Sliding Window) which is the default DPB management in H264 Decoders.

I.3 Comparison of Referencing scheme

Short Term Referencing	Long Term Referencing
DPB management done using Sliding window	DPB management done using MMCO Commands
The overall DPB buffer requirement at the decoder end will be higher	More efficient in DPB buffer requirement
The temporal layers cannot be identified unless informed through SVC syntax or though some external means.	LongTermIndex (LT) can be used to identify the various temporal layers in absence of SVC syntax
The decoding technique is relatively simpler.	The decoding technique is relatively complex

Appendix J

Mapping of Encoding Presets

J.1 Description:

User provided extended parameters of interface structure are taken in account only when encodingPreset is XDM_USER_DEFINED. When encodingPreset is not XDM_USER_DEFINED then encoder selects the appropriate values for extended parameter as per the encodingPreset selected. This appendix explains the extended parameters values that user need to set to meet the exact behavior of a particular encodingPreset like XDM_HIGH_SPEED, XDM_MED_SPEED_HIGH_QUALITY etc. This table is useful when user wants to change a particular parameter without modifying the other default values for a particular preset.

	encodingPreset =XDM_MED_SPEED_HIGH_QUALITY	**encodingPreset =XDM_USER_DEFINED**
rateControlParams.rateControlPreset	IH264_RATECONTROLPARAMS_DEFAULT	IH264_RATECONTROLPARAMS_DEFAULT
intercodingParams.interCodingPreset	IH264_INTERCODING_DEFAULT	IH264_INTERCODING_MED_SPEED_HIGH_QUALITY
intraCodingParams.intraCodingPreset	IH264_INTRACODING_DEFAULT	IH264_INTRACODING_DEFAULT
nalUnitControlParams.nalUnitCodingPreset	IH264_NALU_CONTROL_DEFAULT	IH264_NALU_CONTROL_DEFAULT
sliceCodingParams.sliceCodingPreset	IH264_SLICECODING_DEFAULT	IH264_SLICECODING_DEFAULT
loopFilterParams.loopFilterPreset	IH264_LOOPFILTER_DEFAULT	IH264_LOOPFILTER_DEFAULT
fmoCodingParams.fmoCodingPreset	IH264_FMOCODING_DEFAULT	IH264_FMOCODING_DEFAULT
vuiCodingParams.vuiCodingPreset	IH264_VUICODING_DEFAULT	IH264_VUICODING_DEFAULT
stereoInfoParams.stereoInfoPreset	IH264_STEREOINFO_DISABLE	IH264_STEREOINFO_DISABLE
framePackingSEIParams.framePackingSEIPreset	IH264_FRAMEPACK_SEI_DISABLE	IH264_FRAMEPACK_SEI_DISABLE
svcCodingParams. svcExtensionFlag	IH264_SVC_EXTENSION_FLAG_DISABLE	IH264_SVC_EXTENSION_FLAG_DISABLE
interlaceCodingType	IH264_INTERLACE_DEFAULT	IH264_INTERLACE_DEFAULT
bottomFieldIntra	0	0
gopStructure	IH264ENC_GOPSTRUCTURE_DEFAULT	IH264ENC_GOPSTRUCTURE_DEFAULT
entropyCodingMode	IH264_ENTROPYCODING_DEFAULT	IH264_ENTROPYCODING_DEFAULT
transformBlockSize	IH264_TRANSFORM_DEFAULT	IH264_TRANSFORM_DEFAULT

log2MaxFNumMinus4	10	10
picOrderCountType	IH264_POC_TYPE_DEFAULT	IH264_POC_TYPE_DEFAULT
enableWatermark	0	0
IDRFrameInterval	0	0
pConstantMemory	NULL	NULL
maxIntraFrameInterval	0x7FFFFFFF	0x7FFFFFFF
debugTraceLevel	0	0
lastNFramesToLog	0	0
enableAnalyticinfo	0	0
enableGMVSei	0	0
constraintSetFlags	0	0
enableRCDO	0	0
enableLongTermRefFrame	0	0
LTRPPeriod	0	0
numTemporalLayer	IH264_TEMPORAL_LAYERS_1	IH264_TEMPORAL_LAYERS_1
referencePicMarking	IH264_LONG_TERM_PICTURE	IH264_LONG_TERM_PICTURE

	encodingPreset =XDM_HIGH_SPEED	**encodingPreset =XDM_USER_DEFINED**
rateControlParams.rateControlPreset	IH264_RATECONTROLPARAMS_DEFAULT	IH264_RATECONTROLPARAMS_DEFAULT
intercodingParams.interCodingPreset	IH264_INTERCODING_DEFAULT	IH264_INTERCODING_HIGH_SPEED
intraCodingParams.intraCodingPreset	IH264_INTRACODING_DEFAULT	IH264_INTRACODING_HIGH_SPEED
nalUnitControlParams.nalUnitCodingPreset	IH264_NALU_CONTROL_DEFAULT	IH264_NALU_CONTROL_DEFAULT
sliceCodingParams.sliceCodingPreset	IH264_SLICECODING_DEFAULT	IH264_SLICECODING_DEFAULT
loopFilterParams.loopFilterPreset	IH264_LOOPFILTER_DEFAULT	IH264_LOOPFILTER_DEFAULT
fmoCodingParams.fmoCodingPreset	IH264_FMOCODING_DEFAULT	IH264_FMOCODING_DEFAULT
vuiCodingParams.vuiCodingPreset	IH264_VUICODING_DEFAULT	IH264_VUICODING_DEFAULT
stereoInfoParams.stereoInfoPreset	IH264_STEREOINFO_DISABLE	IH264_STEREOINFO_DISABLE
framePackingSEIParams.framePackingSEIPreset	IH264_FRAMEPACK_SEI_DISABLE	IH264_FRAMEPACK_SEI_DISABLE
svcCodingParams. svcExtensionFlag	IH264_SVC_EXTENSION_FLAG_DISABLE	IH264_SVC_EXTENSION_FLAG_DISABLE
interlaceCodingType	IH264_INTERLACE_DEFAULT	IH264_INTERLACE_DEFAULT
bottomFieldIntra	0	0
gopStructure	IH264ENC_GOPSTRUCTURE_DEFAULT	IH264ENC_GOPSTRUCTURE_DEFAULT
entropyCodingMode	IH264_ENTROPYCODING_DEFAULT	IH264_ENTROPYCODING_DEFAULT
transformBlockSize	IH264_TRANSFORM_DEFAULT	IH264_TRANSFORM_DEFAULT
log2MaxFNumMinus4	10	10

picOrderCountType	IH264_POC_TYPE_DEFAULT	IH264_POC_TYPE_DEFAULT
enableWatermark	0	0
IDRFrameInterval	0	0
pConstantMemory	NULL	NULL
maxIntraFrameInterval	0x7FFFFFFF	0x7FFFFFFF
debugTraceLevel	0	0
lastNFramesToLog	0	0
enableAnalyticinfo	0	0
enableGMVSei	0	0
constraintSetFlags	0	0
enableRCDO	0	0
enableLongTermRefFrame	0	0
LTRPPeriod	0	0
numTemporalLayer	IH264_TEMPORAL_LAYERS_1	IH264_TEMPORAL_LAYERS_1
referencePicMarking	IH264_LONG_TERM_PICTURE	IH264_LONG_TERM_PICTURE

Table 29 Prameter Mapping for various encoding presets

Note :

- encodingPreset as XDM_HIGH_SPEED is only supprted in High profile mode.
- When encoding preset is XDM_HIGH_SPEED then IH264ENC_IntraCodingParams :: intraRefreshMethod is ignored.
- When encoding preset is XDM_HIGH_SPEED then IH264ENC_RateControlParams :: enablePartialFrameSkip is ignored.

This page is intentionally left blank

Appendix K

Region of Interest Encoding

This appendix explains the usage details of Region of Interest (ROI) support.

K.1 Description

Region of Interest (ROI) encoding allows encoding of specified regions in a frame with higher quality as compared to other regions (“regions of non-interest”). This approach helps to maximize the perceptual quality in ROIs and reduce overall bit-rate for the same perceived quality.

K.2 Usage of ROI feature

K.2.1 Enabling ROI Support

The following parameter is added in *IH264ENC_DynamicParams* for enabling ROI support.

XDAS_Int32 enableROI;

Set it to non-zero value to enable ROI encoding. Set it to 0 to disable ROI encoding. Default value is 0 (i.e., ROI is disabled).

ROI will be automatically disabled in case of full frame skip, and for skip macroblocks.

The following parameter has been added in *IH264ENC_Status* so that application can get to know the status of enableROI parameter used for encoding.

XDAS_Int8 enableROI;

If the value of this variable is zero, then ROI is disabled. Otherwise, ROI is enabled.

K.2.2 Interface to Encoder

IH264ENC_RoiInput

‖ Description

IH264ENC_RoiInput parameter is added in *IH264ENC_InArgs* to enable the application to pass ROI-related information to the encoder.

‖ Fields

Field	**Data Type**	**Input/ Output**	**Description**
`listROI[IH264ENC_MAX_ROI]`	`XDM_Rect`	Input	The location of each ROI in terms of top left and bottom right (x,y) co-ordinates. The encoder supports a maximum of 36 ROIs in a frame i.e., IH264ENC_MAX_ROI is 36.

Field	Data Type	Input/ Output	Description
roiType[IH264ENC_MAX_ROI]	XDAS_Int8	Input	Type of each ROI. The supported types are FACE_OBJECT, BACKGROUND_OBJECT, FOREGROUND_OBJECT,DEFAULT_OBJECT, PRIVACY_MASK
numOfROI	XDAS_Int8	Input	Number of ROIs in the current frame.
roiPriority[IH264ENC_MAX_ROI]	XDAS_Int32	Input	Priority information of each ROI. Valid values include all integers between -8 and 8, inclusive. A higher value means that more importance will be given to the ROI compared to other regions. In other words, it determines the number of bits given to ROI. If the ROI type is PRIVACY_MASK then the mask details are provided through this parameter. If ROI is enabled in fixed Qp mode, This filed holds the Qp of specified ROI.

Note:

- If the ROI is detected as `FACE_OBJECT`, then a guard band is added around it. For all other ROI types, no guard band is added.
- If the `roiType` is set as `PRIVACY_MASK` then `roiPriority` will specify the color of mask in 32 bits as mentioned below

  Bits 0-7 : Cb value

  Bits 8-15 : Cr value

  Bits 16-31: Luma Value

  Default value is “0” for GRAY color.

  In case of Multiple masking regions in a frame, all masking regions will use only the color of first masking region by ignoring other regions colors given by user.
- In case of ROI + Multiple Privacy Masking scenario, privacy masking region will be given first follwed by other ROI regions.
- In case of overlapping of ROIs with same type and different priority, the priority used for the overlapped ROI will be the latest priority given by the user.
  - For example, if ROI region 3 (i.e., region corresponding to listROI[2], roiType[2] and roiPriority[2]) and ROI region 5 (i.e., region corresponding to listROI[4], roiType[4] and roiPriority[4]) overlap, the value specified through roiPriority[4] will be used to encode the overlapping MBs.

Appendix L

Watermarking SEI Message

L.1 Brief Description

With the rapid development of Internet technology, media data are used more and more widely. This makes media data not only easy to be transmitted, but also easy to be copied and spread out. Thus, the legal issue arises that some media data should be protected from unauthorized users or operations.

Watermarking is a mechanism to add identity to a bitstream to help decoder to identify the media content. For video security applications, it has become a de-facto requirement to prevent tampering with video.

HDVICP2 H.264 codecs support a watermarking scheme at no loss in performance.

The proposed watermarking mechanism in HDVICP2 H.264 codec is shown in the following figure.

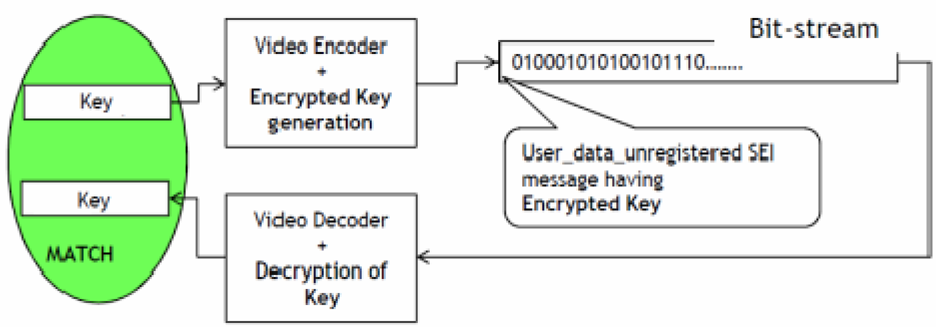


Figure L-1. WaterMarking Mechanism

H.264 Encoder

Encoder accepts a 32-bit key

Encoder encrypts the key using the properties of bit-stream which can be obtained on decoder side as well

Encrypted Key = fn(input_key, bit-stream parameters)

Encoder inserts the encrypted key in the form of user data unregistered SEI message in the encoded stream

H.264 Decoder

Decodes the encrypted key in the form of user data unregistered SEI message

Decrypts the key using the properties of bit-stream

Provides the decrypted 32-bit key

System

Feeds the key on encoder, gets the key from decoder and compares them to identify content tampering

> **Note**
>
> If the system has TI provided H.264 decoder on HDVICP2, it has capability to decrypt the key.
> For the system which doesn't use H.264 decoder from TI, the decoder needs to dump out few properties of bit-stream to decrypt the key

L.2 Usage of watermarking feature

L.2.1 Enabling Watermark Support

An additional create time flag has been added to enable watermarking feature. The following parameter has been added to IH264ENC_Params.

`enableWatermark`	Set to non-zero value to enable encrypting of watermark input key. Watermarking would be disabled if set to 0.

L.2.2 Passing the input key to the Encoder

The following parameter has been added to IH264ENC_InArgs.

`inputKey`	32-bit input key. If the fields of a frame are encoded in separate process calls, the input key fed along with the later process call would be used.

The encrypted key is inserted as part of a user data unregistered SEI message in the next process call before encoding the next frame. In case of interlaced field coding, the encrypted key is computed and inserted at the end of the second field. Only one encrypted key is inserted for a pair of fields.

For the last frame, the watermark SEI message would be inserted at the end of MB data.

> Note:
>
> - In case of interlaced, only the properties of the second field (in decoding order) are considered.
> - The number of bytes encoded for FillerDataBytes_NALU are not accounted in the Total Bytes Generated (TBG)
> - If the XDM_EncMode of the process call is XDM_GENERATE_HEADER, then the inputKey fed in this particular

	process call is ignored by the encoder.

L.3 Watermarking utilization with non-TI decoder

There are lot of applications, which might not use TI provided HDVICP2 based H.264 decoder and need to decrypt the key. This section helps to clarify the method of decryption for such situations

L.3.1 Stream format for Encrypted Key

The encoder inserts the encrypted key in form of user data unregistered SEI message in the encoded stream. The user data unregistered SEI message is identified as watermark message by comparing the first 16 bytes of data with the hexadecimal pattern “e88345e0-61ce-11e1-9ca8-0002a5d5c51b”. If the SEI data matches with the string, we take next 4 bytes of data as watermark encrypted key. Below figure shows the bit-stream content of SEI message. In this figure the K3_K2_K1_K0 is the encrypted key being transmitted as part of SEI message.

Bit Position: 31	24	16	8	0	
0x01	0x00	0x00	0x00	0x00	Base Address of User Defined SEI
0xE8	0x14	0x05	0x06	0x04	
0x61	0xE0	0x45	0x83	0x08	
0x9C	0xE1	0x11	0xCE	0x0C	
0xA5	0x02	0x00	0xA8	0x10	
K0	0x1B	0xC5	0xD5	0x14	
	K3	K2	K1	0x18	
				0x1C	

By using this information, decoder should be able to decode the encrypted key as part of bit stream. Now user has to decrypt the key.

L.3.2 Decyption of Encrypted Key

Using the properties of the bit stream like total bytes generated, number of slices, picture type, etc., along with the encrypted key from the bit stream, one can retrieve the watermark input key that was fed to the encoder.

For this, the following method is defined:

```
U32 RetrieveInputKey(
        U32 encryptedKey,
        U32 totalFrameBytes,
        U32 diffIntraToNonIntraMBs,
        U32 numSlices,
        U32 *pSliceSizes,
        U32 poc,
        U32 numMBsInFrame,
        U32 picType
      );
```

Parameter Name	Data type	Input / Output	Description
encryptedKey	XDAS_Int32	Input	The encrypted key parsed out from the SEI message in the encoded stream
totalFrameBytes	XDAS_Int32	Input	Total bytes generated (till end of MB data)
diffIntraToNonIntraMBs	XDAS_Int32	Input	Absolute difference of number of intra MBs and number of non-intra MBs
numSlices	XDAS_Int32	Input	Number of slices
pSliceSizes	XDAS_Int32	Input	Pointer to an array containing the number of MBs in each slice
Poc	XDAS_Int32	Input	Picture Order Count
numMBsInFrame	XDAS_Int32	Input	Number of MBs in a frame
picType	XDAS_Int32	Input	Picture Type (values used correspond to Table 7-6 in the H.264 standard document)

Note

All the above statistics are accumulated over one frame in progressive cases and only of the second field (decoding order) in interlaced cases.

Return Value

32-bit decrypted key is returned. The decrypted key is expected to match with the input key fed to the encoder.

TI will supply implementation of *RetrieveInputKey* in a library form so user can use along with his/her decoder.

Appendix M

N Frame Process Call Support

Encoder can support N frames processing in single process call. In this method user has to provide all the necessary input parameter (like handle, InArgs, outArgs, InBufs, outBuf) for each frame thorough newly defined XDM API for mullti frame process call. This method is useful in reducing the thread overhead at Media Controller. This support can be utilized to encode either N frames from single channel or one-one from N channels. Encoder is fully unawared of the association between the frames.

M.1 Max value of numChannels (N):

Max value of number of frames that can be processed in single process call depends on max input width among all the inputs.

Table 30 : Maximum number of channels supported for various resolutions

InputWidth	**Max number of channels (IPP seq)**
2048	1
1920	2
1280	24
720	24
640	24
352	24
176	24

M.2 Limitations when using N channel frame processing:

Followings are the limitation of N frame process call support

No B frames. IH264ENC_Params :: IVIDENC2_Params :: maxInterFrameInterval should be one.

No DataSync / Low latency feature. IVIDENC2_Params :: outputDataMode/InputDataMOde should be IVIDEO_ENTIREFRAME

No features such as ROI, FramePackSEI and StereoVideoSEI

All channels should have same MV type (1mv or 4mv).

Minimum bit rate support will not be supported.

Insertion of End of stream and End of sequence will not be supported.

Encoding only Header in a process call is not supported i.e., XDM_EncMode sholud not be XDM_GENERATE_HEADER.

In this method HDVICP acquire is done when first frame processing starts and release is done after all the frame processing is finshed. All the acquire and release is done with the last frame handle.

M.3 XDM interface for Multi Channel process call

```
#define IH264ENC_MAX_LENGTH_PROCESS_LIST (24)

typedef struct
{
  IVIDENC2_Handle handle;
  IVIDEO2_BufDesc *inBufs;
  XDM2_BufDesc *outBufs;
  IVIDENC2_InArgs *inArgs;
  IVIDENC2_OutArgs *outArgs;

} IH264ENC_ProcessParams;

typedef struct
{
  XDAS_Int32 numEntries;
  XDAS_Int32 enableErrorCheck;
  IH264ENC_ProcessParams
processParams[IH264ENC_MAX_LENGTH_PROCESS_LIST];
} IH264ENC_ProcessParamsList ;

typedef struct IH264ENC_Fxns
{
    IVIDENC2_Fxns ividenc;
    XDAS_Int32 (*processMulti)
(IH264ENC_ProcessParamsList *processList);
```

```
} IH264ENC_Fxns;
```

New processMulti API has been be defined for this purpose. If for a channel, incorrect parameter is passed then that particular channel will be skipped for encoding and reaming channels will be encoded. And appropriate error bit will be set for skipped channel.

M.4 Steps to achieve N frame processing in single process call

Populate all the input parmeters like (handle, inBufs, outBufs, inArgs, outArgs) for every frame of input data.

Prepare the instance of the data type IH264ENC_ProcessParamsList.

Call the newly defined API processMulti with address of IH264ENC_ProcessParamsList as a single argument.

After return from the multi process call, utilize the information updated by codec for each frame in corresponding outArgs and outBufs.

Backward compatibility is maintiained after supporting N frame process call. Older API for process call can be used for single frame processing in a process call.

This page is intentionally left blank

Appendix N

Rate Control - High Fidelity Variable Bitrate

This appendix provides an insight to the High Fidelity Variable Bitrate (HF-VBR) Rate Control details of the encoder.

N.1 Description

The Rate Control algorithm (RC) in an encoder is required to

Ensure that the overall bits generated is meets the target bit-rate specified to the encoder.

Ensure that the overall perceptual video quality is maximized.

Conventional rate control algorithms are designed to achieve same average bitrate for all durations of the video sequence High Fidelity Variable Bitrate (HF-VBR) Rate control algorithm is a new rate control which adapts the instantaneous bit-rate to change at different times based on the complexity of video at that point in time. HF-VBR reacts to the instantaneous video complexity in the below way

Use higher-than-average bitrate in highly complex segment of video.

Use lower-than-average bitrate when in simple segment of video.

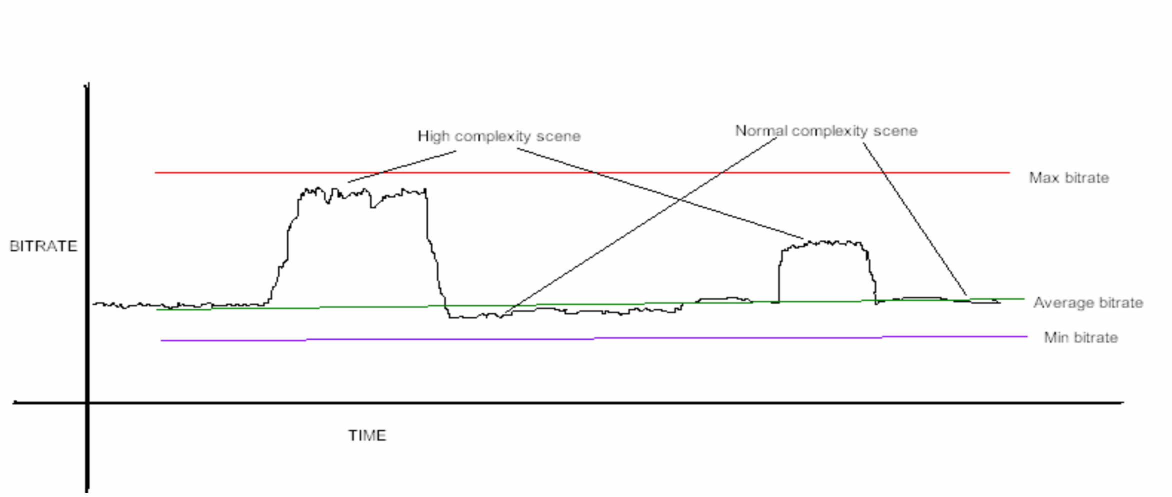


Figure N-1. Graph representing HF-VBR reaction to the video complexity.

HF-VBR rate control allows the bitrate to change based on the complexity of the scene. The rate control takes two inputs viz. 1. targetBitrate and 2. maxBitrate. For scene with normal complexity, the RC operates at targetBitrate. When the scene complexity increases, the RC increases the operating bitrate to a higher value. However it is not allowed to exceed maxBitrate. In a longer duration, the overall bitrate achieved will be targetBitrate. HF-VBR rate control is specially suited for video surveillance where one would intend to encode with better quality when there is an increase in scene complexity

Note: HF-VBR: High Fidelity Variable Bitrate Rate control scheme is designed to achieve targetBitrate in longer duration of time. Hence if one observes the overall bitrate (for long duration), it will always be same as targetBitrate. It will go nearer to maxBitrate for short duration in case the complexity increases.. The complexity estimate done by HF-VBR is based on previous history. If you start and stop the video recording with high complex video thoughout, HF-VBR will not treat it differently. This condition will be treated like normal VBR and you will see the instantaneous bitrate does not go above the targetBitrate for the whole duration. Hence in lab test, one has to be careful when inferring at results. They should let video get recorded with static sequence for few seconds before changing to complex video (like hand movement before camera etc).

N.2 Parameters & Configuration

The below mentioned parameters along with rcAlgo determine the behavior of rate control.

maxBitrate

This parameter limits the maximum bitrate which the rate control can achieve during the high complexity duration of the video. The value of maxBitRate must be at least 1.5 times targetBitRate. Only then HF-VBR: High Fidelity Variable Bitrate control is turned ON in the encoder.

VBRDuration

This parameter is applicable to HF-VBR: High Fidelity Variable Bitrate. The time interval (in seconds) during which encoder collects statistics related to the complexity of the video to vary the instantaneous bitrate. Larger value of this parameter results in the rate control algorithm reacting to complexity changes slowly. Allowed values are 1-3600 only

If VBRDuration is not set and only maxBitrate value is set to a value that is atlaest 1.5 times the target bit rate then VBRDuration is taken by the encoder to be 8sec.

VBRsensitivity

This parameter is applicable to HF-VBR: High Fidelity Variable Bitrate. It controls the sensitivity of the HF-VBR algorithm towards the complexity of video. It can take any value from 0 to 8. A lower value signifies that maxBitrate will be used for very complex scene, in case complexity increase is not high, HF-VBR will choose a bitrate between target bitrate and maxBitrate. If set to higher value say 8, rate control tries to achieve maxBitrate even for small complexity increase. Since complexity is a subjective measure, it is recommended to tune this parameter based on user expectation.

N.3 How to specify RC mode

```
rateControlPreset = IVIDEO_STORAGE (2)
```

or

```
rateControlPreset = IVIDEO_USER_DEFINED (5)
rcAlgo = IH264_RATECONTROL_PRC (0)
```

and

```
maxBitRate >= (1.5 x targetBitRate )
```

This page is intentionally left blank

Appendix O

Gradual Decoder Refresh (GDR)- an Error resilience Feature

This appendix provides a brief understanding of GDR and explains the usage details of GDR.

O.1 Description

GDR is a mechanism, which creates a bit-stream having spatially coded macroblocks (intra macroblocks) in moving overlapped (or non-overlapped) region of the picture. These intra macrblocks should cover the entire picture region over few pictures.

O.1.1 There are two schemes in GDR

Vertical GDR

Horizontal GDR

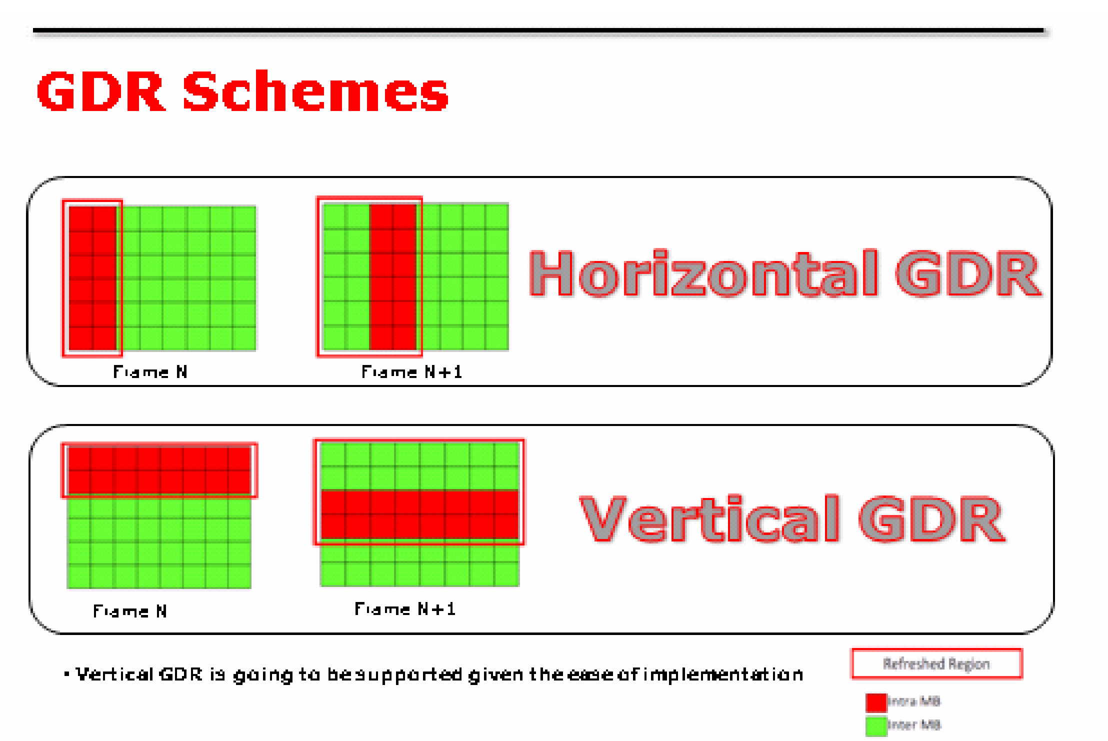


Figure O-1. GDR Schemes.

Vertical GDR is implemented in HDVICP2 H.264 encoder.

Refreshed region is the portion which is error free in a given picture. In above figure, refreshed region is outlined with red box. The number of pictures required to refresh the decoder are mostly referred as GDR period. In above figure horizontal GDR has GDR Period of 4 and Veritcal GDR has GDR Period of 3. Above figure shows, non overlapped GDR scheme, but one can also have overlapped GRD scheme in which the intra region of Frame N and Frame N+1 overlaps.

O.1.2 Constraints on Intra and Inter MBs in a GDR'd region

In Refreshed Region intra and inter MBs need to have below constraints

Intra MBs should only refer to spatial region which is part of refreshed region

Inter MBs should only refer the temporal region which is part of refreshed region in that picture

O.2 Parameters and Configuration

O.2.1 Enabling GDR feature

```
Set intraCodingPreset as IH264_INTRACODING_USERDEFINED (1) and
Set intraRefreshMethod as IH264_INTRAREFRESH_GDR (4)
This method of intra refresh mechanism is not supported for 1)interlaced
sequences and 2) 'B' frame cases.
User can set GDR to be started at any picture during encoding session. To
start GDR, user need to set below parameter
IVIDENC2_InArgs::control = IH264ENC_CTRL_START_GDR
```

O.2.2 GDR control Parameters

Below are the contorl parameter to allow user to set GDR period and the control for overlapped portion

intraRefreshRate	This parameter is treated/interpreted as the number of rows to be intra refreshed per frame. Supported values are [0, NonZero]
gdrOverlapRows	This parameter is the number of rows overlap between successive GDR frames and value should be less than intraRefreshRate. Supported values are [0, NonZero].

Based on network condition/feedback Encoder would be commanded through interface to start GDR at any frame