H.264 Base/Main/High Profile Encoder on DM365/DM368

User's Guide

Literature Number: SPRUEU9C
August 2010

IMPORTANT NOTICE

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

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

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

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

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

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

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

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
Wireless	www.ti.com/wireless-apps		

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

Read This First

About This Manual

This document describes how to install and work with Texas Instruments' (TI) H.264 Base/Main/High Profile Encoder implementation on the DM365/DM368 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) and IRES standards. XDM and IRES are extensions of 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 DM365/DM368 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, Frame work Components, and software architecture. 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.
- **Appendix A – Time-Stamp Insertion**, describes insertion of frame time-stamp through the Supplemental Enhancement Information (SEI) Picture Timing message.

- **Appendix B – Error Description**, provides a list of error descriptions.
- **Appendix C – VICP buffer usage by codec**, provides details of how VICP buffers are used by codec.
- **Appendix D – ARM926 TCM buffer usage by codec**, provides details of using ARM926 TCM buffer by codec.
- **Appendix E - Simple Two-pass Encoding Sample Usage,** explains how multi-pass encoding can be used to improve the quality of the H264 encoded video
- **Appendix F – Revision History,** highlights the changes made to the SPRUEU9A codec specific user guide to make it SPRUEU9B.

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* (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* (SPRU360) describes all the APIs that are defined by the TMS320 DSP Algorithm Interoperability Standard (also known as XDAIS) specification.
- *Using IRES and RMAN Framework Components for C64x+* (literature number SPRAAI5) provides an overview of the IRES interface, along with some concrete resource types and resource managers that illustrate the definition, management and use of new types of resources.

Related Documentation

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

- ISO/IEC 14496-10:2005 (E) Rec. H.264 (E) ITU-T Recommendation

Abbreviations

The following abbreviations are used in this document.

Table 1-1. List of Abbreviations

Abbreviation	Description
ASO	Arbitrary Slice Ordering
AVC	Advanced Video Coding

Abbreviation	Description
BIOS	TI's simple RTOS for DSPs
CAVLC	Context Adaptive Variable Length Coding
CABAC	Context Adaptive Binary Arithmetic Coding
D1	720x480 or 720x576 resolutions in progressive scan
DCT	Discrete Cosine Transform
DDR	Double Data Rate
DMA	Direct Memory Access
FC	Framework components
FMO	Flexible Macro-block Ordering
HD 720 or 720p	1280x720 resolution in progressive scan
HDTV	High Definition Television
HDVICP	High Definition Video and Imaging Co-processor sub-system
IDR	Instantaneous Decoding Refresh
ITU-T	International Telecommunication Union
JM	Joint Menu
JVT	Joint Video Team
MB	Macro Block
MBAFF	Macro Block Adaptive Field Frame
MJCP	MPEG JPEG Co-Processor
MPEG	Motion Pictures Expert Group
MV	Motion Vector
NAL	Network Adaptation Layer
NTSC	National Television Standards Committee
PDM	Parallel Debug Manager
PicAFF	Picture Adaptive Field Frame
PMP	Portable Media Player
PPS	Picture Parameter Set

Abbreviation	Description
PRC	Perceptual Rate Control
RTOS	Real Time Operating System
RMAN	Resource Manager
SEI	Supplemental Enhancement Information
SPS	Sequence Parameter Set
VGA	Video Graphics Array
VICP	Video and Imaging Co-Processor
XDAIS	eXpressDSP Algorithm Interface Standard
XDM	eXpressDSP Digital Media
YUV	Color space in luminance and chrominance form
ROI	Region Of Interest
STP	Simple Two Pass

Note:

MJCP and VICP refer to the same hardware co-processor blocks.

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 Base/Main/High Profile Encoder on DM365/DM368) 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.

Contents

Figures

This page is intentionally left blank

Tables

This page is intentionally left blank

Chapter 1

Introduction

This chapter provides a brief introduction to XDAIS, XDM, and DM365 software architecture. It also provides an overview of TI's implementation of the H.264 Base/Main/High Profile Encoder on the DM365/DM368 platform and its supported features.

Topic **Page**

1.1 Software Architecture

DM365/DM368 codec provides XDM compliant API to the application for easy integration and management. The details of the interface are provided in the subsequent sections.

DM365/DM368 is a digital multi-media system on-chip primarily used for video security, video conferencing, PMP and other related application.

DM365/DM368 codec are OS agonistic and interacts with the kernel through the Framework Component (FC) APIs. FC acts as a software interface between the OS and the codec. FC manages resources and memory by interacting with kernel through predefined APIs.

Following diagram shows the software architecture.

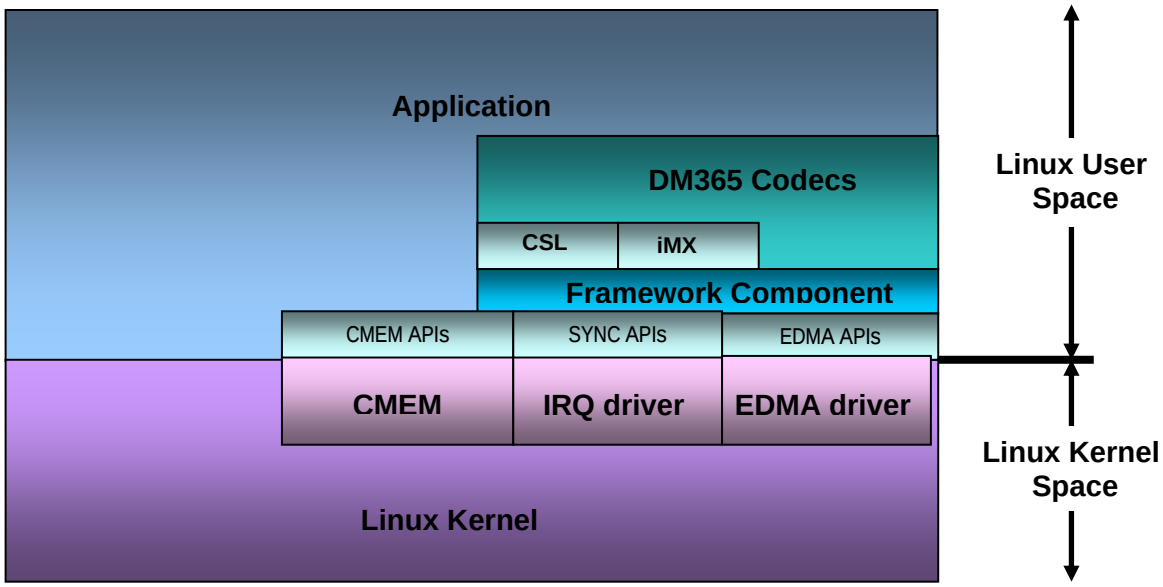


Figure 1-1. Software Architecture.

1.2 Overview of XDAIS, XDM, and Framework Component Tools

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 a TMS320 DSP Algorithm Standard (xDAIS) interface for management and utilization of special resource types such as hardware accelerators, certain types of memory and DMA. RMAN is a generic Resource Manager that manages software component's logical resources based on their IRES interface configuration. Both IRES and RMAN are Framework Component modules.

1.2.1 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 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 two more optional APIs `algNumAlloc()` and `algMoved()`. For more details on these APIs, see *TMS320 DSP Algorithm Standard API Reference* (SPRU360).

1.2.2 XDM Overview

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

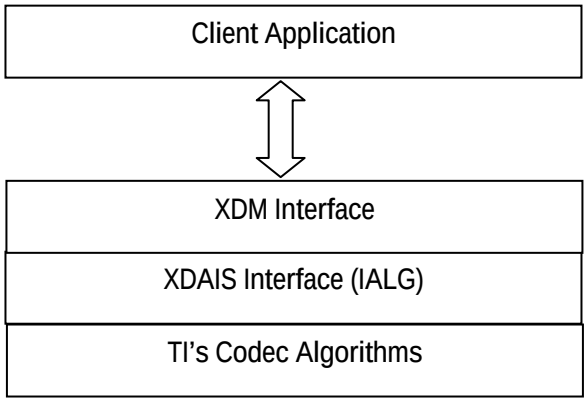
- `control()`
- `process()`

The `control()` API provides a standard way to control an algorithm instance and receive status information from the algorithm in real-time. The `control()` API replaces the `algControl()` API defined as part of the IALG interface. The `process()` API does the basic processing (encode/decode) of data. This API represents a blocking call for the encoder and the decoder, that is, with the usage of this API, the control is returned to the calling application only after encode or decode of one unit (frame) is completed. Since in case of DM365/DM368, the main encode or decode is carried out by the hardware accelerators, the host processor

from which the `process()` call is made can be used by the application in parallel with the encode or the decode operation. To enable this, the framework provides flexibility to the application to pend the encoder task when the frame level computation is happening on coprocessor.

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

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



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

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

1.2.3 Framework Component

As discussed earlier, Framework Component acts like a middle layer between the codec and OS and also serves as a resource manager. The following block diagram shows the FC components and their interfacing structure.

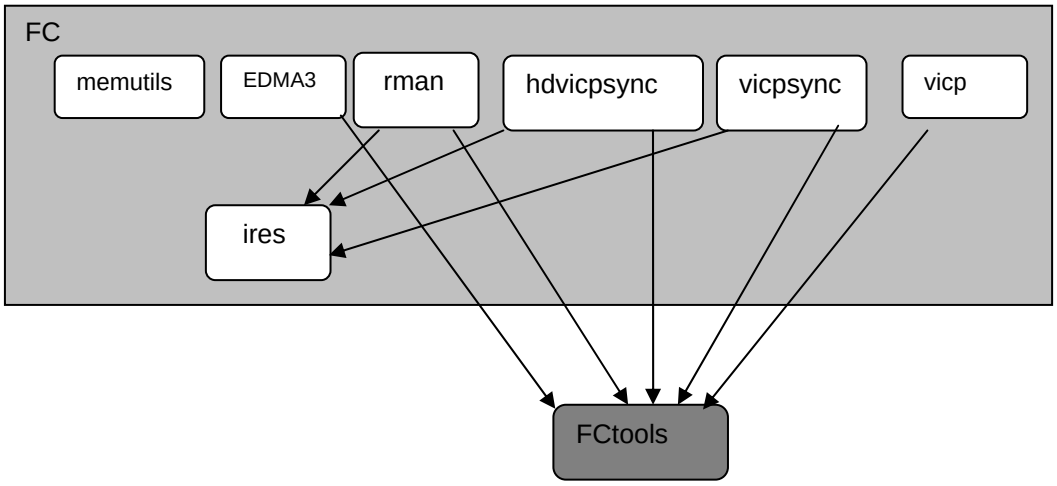


Figure 1-2. Framework Component Interfacing Structure.

Each component is explained in detail in the following sections.

1.2.3.1 IRES and RMAN

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).

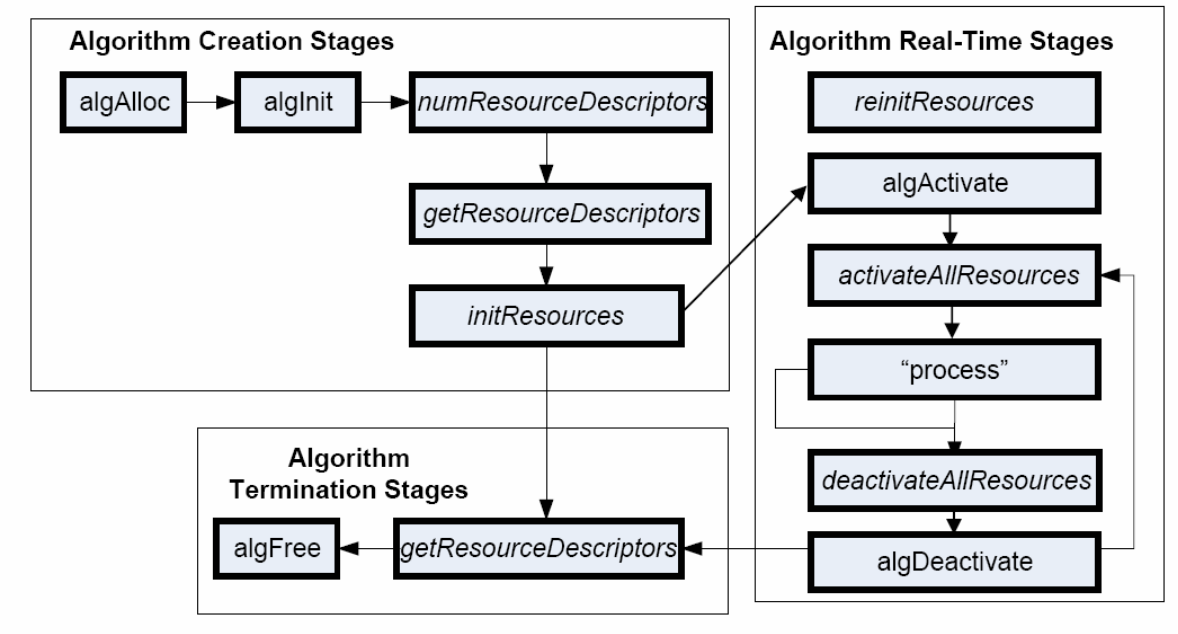


Figure 1-3. IRES Interface Definition and Function-calling Sequence.

In DM365/DM368, FC manages multiple resources for smooth interaction with other algorithms and application. The resources and the utilities provided by FC are listed in this section.

1.2.3.2 HDVICP

The IRES HDVICP Resource Interface, IRES_HDVICP, allows algorithms to request and receive handles representing Hardware Accelerator resource, HDVICP, on supported hardware platforms. Algorithms can request and acquire one of the co-processors using a single IRES request descriptor. IRES_HDVICP is an example of a very simple resource type definition, which operates at the granularity of the entire processor and does not publish any details about the resource that is being acquired other than the 'id' of the processor. It leaves it up to the algorithm to manage internals of the resource based on the ID.

1.2.3.3 EDMA3

The IRES EDMA3 Resource Interface, IRES_EDMA3CHAN, allows algorithms to request and receive handles representing EDMA3 resources associated with a single EDMA3 channel. This is a very low-level resource definition.

Note:

The existing xDAIS IDMA3 and IDMA2 interfaces can be used to request logical DMA channels, but the IRES EDMA3CHAN interface provides the ability to request resources with finer precision than with IDMA2 or IDMA3.

1.2.3.4 VICP

VICP resource manager provides access to its VICP compute engine and its buffer. The compute engines are MJCP, NSF, IMX0 and IMX1. In addition to this, the VICP buffers are also assumed as resources and can be requested as either named buffers (for MPEG and JPEG codec operation) of generic scratch buffer (for H.264 codec operation).

1.2.3.5 HDVICP Sync

Synchronization is necessary in a coprocessor system. HDVICP sync provides framework support for synchronization between codec and HDVICP coprocessor usage. This module is used by frameworks or applications, which have xDIAS algorithms that use HDVICP hardware accelarators.

1.2.3.6 Memutils

This is for generic APIs to perform cache and memory related operations.

- `cacheInv` – Invalidates a range of cache
- `cacheWb` – Writes back a range of cache
- `cacheWbInv` – Writes back and invalidate cache
- `getPhysicalAddr` – Obtains physical (hardware specific) address

1.2.3.7 TCM

ARM TCM resource manager provides access to request ARM926 TCM memory. ARM926 in DM365/DM368 has 32K TCM, which can be allocated to codec/algorithm on request. The allocation is in granularity of 1/2K blocks, which can be used as scratch memory by the codec/algorithm.

1.3 Overview of H.264 Base/Main/High Profile Encoder

H.264 (from ITU-T, also called as H.264/AVC) is a popular video coding algorithm enabling high quality multimedia services on a limited bandwidth network. H.264 standard defines several profiles and levels that specify restrictions on the bit stream and hence limits the capabilities needed to decode the bit streams. Each profile specifies a subset of algorithmic features and limits that all decoders conforming to that profile may support. Each level specifies a set of limits on the values that may be used by the syntax elements in the profile.

Some important H.264 profiles and their special features are (These are feature as defined by H.264 standard, few of them may not be part of DM365/DM368 H.264 implementation):

- Baseline Profile:
  - Only I and P type slices are present
  - Only frame mode (progressive) picture types are present
  - Only CAVLC is supported
  - ASO/FMO and redundant slices for error concealment is supported
- High Profile:
  - Only I, P, and B type slices are present
  - Frame and field picture modes (in progressive and interlaced modes) picture types are present
  - Both CAVLC and CABAC are supported
  - ASO is not supported
  - Transform 8x8 is supported
  - Sequence scaling list is supported
  - B frames and weighted prediction.

The input to the encoder is a YUV sequence, which can be of format 420 with the chroma components interleaved in little endian. The output of the encoder is an H.264 encoded bit-stream in the byte-stream syntax. The byte-stream consists of a sequence of byte-stream NAL unit syntax structures. Each byte-stream NAL unit syntax structure contains one start code prefix of size four bytes and value 0x00000001, followed by one NAL unit syntax structure. The encoded frame data is a group of slices, each is encapsulated in NAL units. The slice consists of the following:

- Intra coded data: Spatial prediction mode and prediction error data, subjected to DCT and later quantized.
- Inter coded data: Motion information and residual error data (differential data between two frames), subjected to DCT and later quantized.

The first frame is called Instantaneous Decode Refresh (IDR) picture frame. The decoder at the receiving end reconstructs the frame by spatial intra-prediction specified by the mode and by adding the prediction error. The subsequent frames may be intra or inter coded.

In case of inter coding, the decoder reconstructs the bit-stream by adding the residual error data to the previously decoded image, at the location specified by the motion information. This process is repeated until the entire bit-stream is decoded.

In motion estimation, the encoder searches for the best match in the available reference frame(s). After quantization, contents of some blocks become zero. H.264 Encoder tracks this information and passes the information of coded 4x4 blocks to inverse transform so that it can skip computation for those blocks that contain all zero co-efficients and are not coded.

H.264 Encoder defines in-loop filtering to avoid blocks across the 4x4 block boundaries. It is the second most computational task of H.264 encoding process after motion estimation. In-loop filtering is applied on all 4x4 edges as a post-process and the operations depend upon the edge strength of the particular edge.

H.264 Encoder applies entropy coding methods to use context based adaptivity, which in turn improves the coding performance. All the macro blocks, which belong to a slice, must be encoded in a raster scan order. Baseline profile uses the Context Adaptive Variable Length Coding (CAVLC). CAVLC is the stage where transformed and quantized coefficients are entropy coded using context adaptive table switching across different symbols. The syntax defined by the H.264 Encoder stores the information at 4x4 block level.

The following figure depicts the working of the encoder.

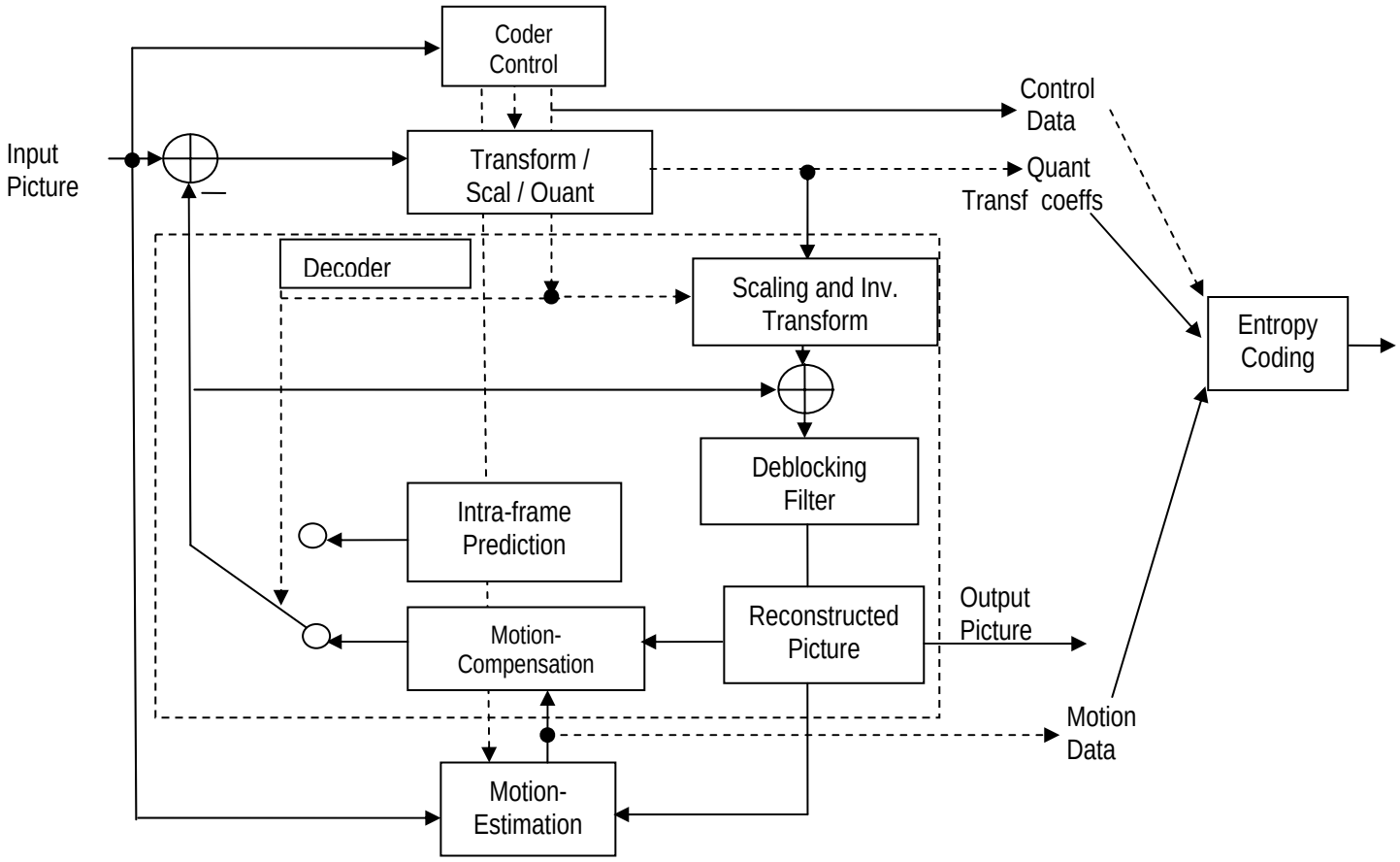


Figure 1-4. Block Diagram of H.264 Encoder.

From this point onwards, all references to H.264 Encoder mean H.264 Base/Main/High Profile Encoder only.

1.4 Supported Services and Features

This user guide accompanies TI's implementation of H.264 Encoder on the DM365/DM368 platform.

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

- eXpressDSP Digital Media (XDM1.0 IVIDENC1) interface compliant
- Compliant with H.264 High Profile up to level 5.0
- Supports resolutions up to 2048x2048
- Supports YUV420 semi planer input format for the frames
- Supports progressive and interlaced encoding
- Generates bit-stream compliant with H.264 standard
- Supports CAVLC and CABAC encoding
- Supports sequence scaling matrix
- Supports transform 8x8 and transform 4x4
- Supports frame based encoding with frame size being multiples of 2
- Supports rate control (CBR and VBR)
- Supports Insertion of Buffering Period and Picture Timing Supplemental Enhancement Information (SEI) and Video Usability Information (VUI)
- Supports Unrestricted Motion Vectors (UMV)
- Supports Half Pel and Quarter Pel Interpolation for motion estimation
- Supports following Smart Codec features:
  - Simple Two Pass (STP) Encoding
  - Region of Interest (ROI)
- Supports Low latency feature
  - Can be configured to provide output at NAL granularity or after entire frame is encoded.
  - Supports encoded output in NAL stream or Bytes stream format

DM365/DM368 H.264 encoder can be configured in two modes:

- Platinum mode, which gives 1080P@30fps performance in DM368 – 432 Mhz device
- Version 1.1 backward compatible mode which gives performance of 720P@30fps on DM365/DM368 - 300 MHz

This version of the encoder does not support the following features as per the Baseline Profile feature set:

- Error Resilience features such as ASO/FMO and redundant slices

- Adaptive Reference Picture Marking
- Reference Picture List Reordering

1.5 Comparison between version 01.10.00.xx with new version 02.00.00.xx (Platinum Encoder)

Version 02.00.00.xx is a new enhanced codec version with 1.5x better performance than earlier version without affecting quality. Few of the enhancements are listed below:

- Achieves 1080P@30fps on DM368.
- More feature rich codecs which includes
  - Smart codec technology
  - Low latency API support

Version 02.00.00.xx also supports version 1.1 standard mode as a backward compatible option. This can be enabled by setting `encodingPreset = XDM_USER_DEFINED` and `encQuality = 0`. It enables application that needs low-resolution encoding, lesser EDMA channels or some specific tools like perceptual rate control.

Feature	Version 1.1 - Gen 1	Version 2.0 - Platinum
Resolution	Min – 128 x 96 Max – 2k x 2k	Min – 320 x 128 Max – Current (2k x 2k)
Performance	720P@30fps on DM365/DM368	1080P@30fps on DM368
EDMA channels	37	46
Support for Ver 1.1 – Gen1	NA	YES

This page is intentionally left blank

Chapter 2

Installation Overview

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

Topic **Page**

2.1 System Requirements for Linux

This section describes the hardware and software requirements for the normal functioning of the codec in MV Linux OS. For details about the version of the tools and software, see Release Note

2.1.1 Hardware

- DM365/DM368 EVM (Set the bits 2 and 3 of switch SW4 to low(0) position and Set the bits 4 and 5 of switch SW5 to high(1) position)
- RS232 cable and network cable

2.1.2 Software

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

- **Build Environment:** This project is built using Linux with MVL ARM tool chain.
- **ARM Tool Chain**: This project is compiled and linked using MVL ARM tool chain.

2.2 Installing the Component for Linux

The codec component is released as a compressed archive. To install the codec, extract the contents of the tar file onto your local hard disk. The tar file extraction creates a directory called dm365_h264enc_xx_xx_xx_xx_production. Figure 2-5 shows the sub-directories created in this directory.

Note:

xx_xx_xx_xx in the directory name is the version of the codec. For example, If the version of the codec is 02.00.01.00, then the directory created on extraction of tar file is dm365_h264enc_02_00_01_00_production.

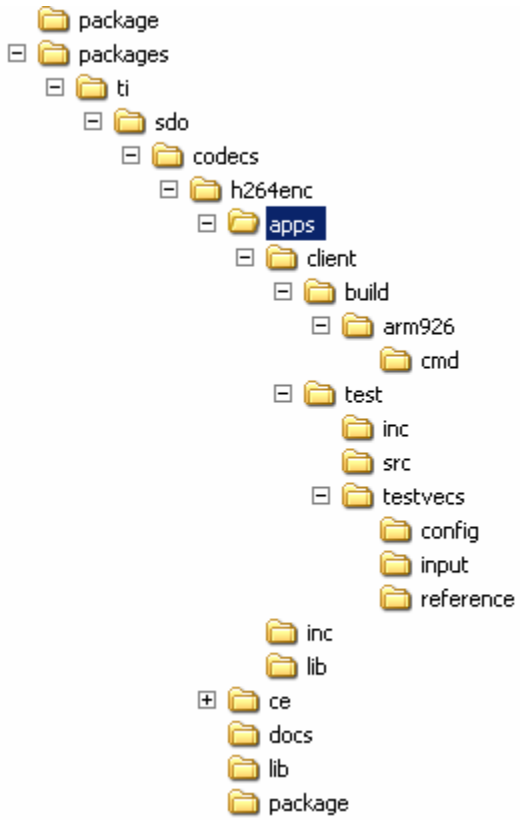


Figure 2-5. Component Directory Structure for Linux.

Table 2-2 provides a description of the sub-directories created in the dm365_h264enc_xx_xx_xx_xx_production directory.

Table 2-2. Component Directories for Linux.

Sub-Directory	Description
\package	Contains files related while building the package
\packages\ti\sdo\codecs\h264enc\lib	Contains the codec library files on host
\packages\ti\sdo\codecs\h264enc\docs	Contains user guide, datasheet, and release notes
\packages\ti\sdo\codecs\h264enc\apps\client\build\arm926	Contains the makefile to built sample test application
\packages\ti\sdo\codecs\h264enc\apps\client\build\arm926\cmd	Contains a template (.xdt) file to used to generate linker command file
\packages\ti\sdo\codecs\h264enc\apps\client\build\arm926\map	Contains the memory map generated on compilation of the code
\packages\ti\sdo\codecs\h264enc\apps\client\test\src	Contains application C files
\packages\ti\sdo\codecs\h264enc\apps\client\test\inc	Contains header files needed for the application code

Sub-Directory	Description
\packages\ti\sdo\codecs\h264enc\apps\client\test\testvecs\input	Contains input test vectors
\packages\ti\sdo\codecs\h264enc\apps\client\test\testvecs\output	Contains output generated by the codec
\packages\ti\sdo\codecs\h264enc\apps\client\test\testvecs\reference	Contains read-only reference output to be used for verifying against codec output
\packages\ti\sdo\codecs\h264enc\apps\client\test\testvecs\config	Contains configuration parameter files

2.3 Building and Running the Sample Test Application on Linux

To build the sample test application in linux environment, follow these steps

1) Verify that dma library h264v_ti_dma_dm365.a exists in the packages\ti\sdo\codecs\h264enc\lib.

2) Verify that codec object library library h264venc_ti_arm926.a exists in the \packages\ti\sdo\codecs\h264enc\lib.

3) Ensure that you have installed the LSP, Montavista arm tool chain, XDC, Framework Components releases with version numbers that are mentioned in the release notes.

4) In the folder \packages\ti\sdo\codecs\h264enc\client\build\arm926, change the paths in the file rules.make according to your setup.

5) Open the command prompt at the sub-directory \packages\ti\sdo\codecs\h264enc\client\build\arm926 and type the command make. This generates an executable file h264venc-r in the same directory.

To run the executable generated from the above steps:

1) Load the kernel modules by typing the command ./loadmodules.sh, which initializes the CMEM pools.

2) Now branch to the directory where the executable is present and type ./h264venc-r in the command window to run.

2.4 Configuration Files

This codec is shipped along with:

- Generic configuration file (testvecs.cfg) – list of configuration files for running the codec on sample test application.
- Encoder configuration file (testparams.cfg) – specifies the configuration parameters used by the test application to configure the Encoder.

2.4.1 Generic Configuration File

The sample test application shipped along with the codec uses the configuration file, Testvecs.cfg for determining the input and reference files for running the codec and checking for compliance. The testvecs.cfg file is available in the \packages\ti\sdo\codecs\h264enc\apps\client\test\testvecs\config sub-directory.

The format of the testvecs.cfg file is:

```
X
config
input
output/reference
recon
```

where:

- `X` may be set as:
  - 1 - for compliance checking, no output file is created
  - 0 - for writing the output to the output file
- `config` is the Encoder configuration file. For details, see Section 2.4.2.
- `input` is the input file name (use complete path).
- `output/reference` is the output file name (if `X` is 0) or reference file name (if `X` is 1) (use complete path).
- `recon` is reconstructed YUV output file name (use complete path).

A sample testvecs.cfg file is as shown:

For output dump mode:

```
0
..\..\..\test\testvecs\config\testparams.cfg
..\..\..\test\testvecs\input\input.yuv
..\..\..\test\testvecs\output\output.264
..\..\..\test\testvecs\output\recon.yuv
```

For reference bit-stream compliance test mode:

```
1
..\..\..\test\testvecs\config\testparams.cfg
..\..\..\test\testvecs\input\input.yuv
..\..\..\test\testvecs\reference\reference.264
..\..\..\test\testvecs\output\recon.yuv
```

2.4.2 Encoder Configuration File

The encoder configuration file, testparams.cfg contains the configuration parameters required for the encoder. The testparams.cfg file is available in the \client\test\testvecs\config sub-directory.

A sample Testparams.cfg file is as shown:

```
# Config File Format is as follows
# <ParameterName> = <ParameterValue> # Comment
########################################################
Parameters
########################################################
ImageWidth = 1280    # Image width in Pels, must be multiple of 16
ImageHeight = 720    # Image height in Pels, must be multiple of 16
FrameRate   = 30000  # Frame Rate per second*1000(1-120)
BitRate    = 4000000 # Bitrate(bps)  #if ZERO=>> RC is OFF
ChromaFormat  = 9    # 9 => XDM_YUV_420P
InterlacedVideo = 0  # 0: Progressive, 1 :Interlaced
TimerScale = 60.     # Timer Resolution for Picture Timing
NumUnitsInTicks = 1  # Number of Timer units per Tick
AspectRatioWidth = 1 # Aspect Ratio Width Scale
AspectRatioHeight = 1 # Aspect Ratio Height Scale
PixelRange      = 1   # 1 =>Y- 0 to 255, Cb/Cr-0 to 255
                        0 => Y-16 to 235, Cb/Cr-16 to 240
EnableVUIParam  = 1   # 1 => Enable VUI parameters,
0 => Disable VUI Parameters
EnableBufSEI    =   1   # 1 => Enable Buffering Period SEI Message,
                          0 => Disable
ME_Type          = 0   # ME search algorithm
                         0 => Normal,
                          1 => Low Power
RC_PRESET        = 5   # 1 => Low Delay,
                         2 => Storage,
                          3 => 2 Pass,
                          4 => None,
                          5 => user defined
ENC_PRESET       = 3   # 3 => User Defined Parameters
########################################################
# Encoder Control
########################################################
ProfileIDC      = 100     # Profile IDC (66=baseline, 77=main,
                          100=high profile)
```

```
LevelIDC     = 30        # Level IDC (e.g. 20 = level 2.0)
IntraPeriod     = 30     # Period of I-Frames
IDRFramePeriod  = 0      # Period of IDR Frames
FramesToEncode  = 10     # Number of frames to be coded
SliceSize  = 0           # Size of each slice
EnMeMultiPart   = 0      # 1 => Enable MB Partitions,
                             0=> Single MV for each MB
RateControl     = 1      # 0 => CBR,
                             1 => VBR,
                          2 = Fixed QP
MaxDelay        = 2000   # Delay Parameter for Rate Control in
                            Milliseconds
QPInit          = 28     # Initial QP for RC (-1,0-51)
QPISlice        = 48     # Quant. param for I Slices (0-51)
QPSlice          = 48    # Quant. param for non - I slices (0-51)
MaxQP           = 42     # Maximum value for QP (0-51)
MinQP           = 0      # Minimum value for QP (0-51)
MaxQPI           = 40    # Maximum value for QP for I frame(0-51)
MinQPI           = 0     # Minimum value for QP for I Frame(0-51)
IntraThrQF      = 0      # Reserved
AirRate         = 0      # Number of Forced Intra MBs
UnRestrictedMV  = 0      #1: Enable 0:Disable
EntropyCodingMode = 1    # Entropy Coding Mode (0 = CAVLC, 1 = CABAC)
Transform8x8FlagIntra = 1  # 0 = Disable,
                           #  1 = Enable
Transform8x8FlagInter = 1  # 0 = Disable,
                           #  1 = Enable
SequenceScalingFlag = 0    # 0 = Disable,
                           #  1 = Auto,
                           #  2 = Low,
                           #  3 = Moderate,
                           #  4 = Reserved
PerceptualRC    = 1        # 1 => Enable Perceptual QP modulation,
                           #  0 => Disable
EncoderQuality  = 1        # 0 => Ver 1.1 mode (Backward compatible),
                           # 2 => Platinum mode
mvSADout = 0               # 0=>disable mvsad out,
                           # 1=>enable mvsad out
useARM926Tcm = 1           # 0->do not use arm 926 tcm,
                           # 1-> use arm 926 tcm
enableROI = 0              # 0->disable ROI
                           # 1-> enable ROI
mapIMCOPtoDDR = 0          #0->do not use DDR
                           # 1-> use DDR instead of IMCOP
metaDataGenerateConsume = 0 # 0->Not in use,
                             # 1-> Generate Meta data,
                             # 2-> Use Metadata generated by other encoder.
sliceMode = 0           # 0 -> no multiple slices,
                        # 1 -> Reserved,
                        # 2 -> multiple slices-MBs/slice,
                        # 3 -> multiple slices - Rows/slice
outputDataMode = 1       # 0 -> low latency, encoded streams produced
                         # after N (configurable) slices encode,
                         # 1 -> encoded stream produce at the end of frame
sliceFormat = 1          # 0-> encoded stream in NAL unit format,
                         #1 -> encoded stream in bytes stream format


#######################################################
Loop filter parameters
#######################################################
LoopFilterDisable = 0   # Disable loop filter in slice header
                           0=Filter,
```

```
                              1=No Filter,
                              2 = Disable across Slice Boundaries
```

To check the functionality of the codec for the inputs other than those provided with the release, change the configuration file accordingly, and follow the steps as described in Section 2.2.

2.4.3 Encoder Sample Base Param Setting

The encoder can be run in `IVIDENC1` base class setting. The extended parameter variables of encoder will then assume default values. The following list provides the typical values of `IVIDENC1` base class variables.

```
typedef struct IVIDENC1_Params {
XDAS_Int32 size;
XDAS_Int32 encodingPreset = XDM_HIGH_SPEED; // Value = 2
XDAS_Int32 rateControlPreset = IVIDEO_STORAGE; //value = 2
XDAS_Int32 maxHeight =  720;
XDAS_Int32 maxWidth =  1280;
XDAS_Int32 maxFrameRate = 120000;
XDAS_Int32 maxBitRate = 50000000;
XDAS_Int32 dataEndianness = XDM_BYTE;
XDAS_Int32 maxInterFrameInterval = 1;
XDAS_Int32 inputChromaFormat = XDM_YUV_420SP;  //value = 9
XDAS_Int32 inputContentType = IVIDEO_PROGRESSIVE;
XDAS_Int32 reconChromaFormat  XDM_YUV_420SP;  //value = 9;
} IVIDENC1_Params;
typedef struct IVIDENC1_DynamicParams {
XDAS_Int32 size;            /**< @sizeField */
XDAS_Int32 inputHeight;     /**< Input frame height. */
XDAS_Int32 inputWidth;      /**< Input frame width. */
XDAS_Int32 refFrameRate = 30000;
XDAS_Int32 targetFrameRate  = 30000;
XDAS_Int32 targetBitRate;  < 10000000 /**< Target bit rate
in bits per second. */
XDAS_Int32 intraFrameInterval = 29;
XDAS_Int32 generateHeader = 0;
XDAS_Int32 captureWidth; // for demo, same as inputWith
XDAS_Int32 forceFrame;  = IVIDEO_NA_FRAME
XDAS_Int32 interFrameInterval = 0;
XDAS_Int32 mbDataFlag = 0;
} IVIDENC1_DynamicParams;
typedef struct IVIDENC1_InArgs {
XDAS_Int32 size;            /**< @sizeField */
XDAS_Int32 inputID;  /* as per application*/
XDAS_Int32 topFieldFirstFlag = 0;
} IVIDENC1_InArgs;
```

2.5 Standards Conformance and User-Defined Inputs

To check the reference bit-stream conformance of the codec for the default input file shipped along with the codec, follow the steps as described in Section 2.3.

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\input sub-directory.

2) Copy the reference files to the \client\test\testvecs\reference sub-directory.

3) Edit the configuration file, Testvecs.cfg available in the \client\test\testvecs\config sub-directory. For details on the format of the testvecs.cfg file, see section 2.4.

   For each encoded frame, the application displays the message indicating the frame number. In reference bit-stream compliance check mode, the application additionally displays FAIL message, if the bit-stream does not match with reference bit-stream.

   After the encoding is complete, the application displays a summary of total number of frames encoded. In reference bit-stream compliance check mode, the application additionally displays PASS message, if the bit-stream matches with the reference bit-stream.

   If you have chosen the option to write to an output file (X is 0), you can use any of the standard file comparison utility to compare the codec output with the reference output and check for conformance.

2.6 Uninstalling the Component

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

This page is intentionally left blank

Chapter 3

Sample Usage

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

Topic **Page**

3.1 Overview of the Test Application

The test application exercises the `IVIDENC1` base class of the H.264 Encoder library. The main test application files are h264encoderapp.c and h264encoderapp.h. These files are available in the \client\test\src and \client\test\inc sub-directories respectively.

Figure 3-1 depicts the sequence of APIs exercised in the sample test application.

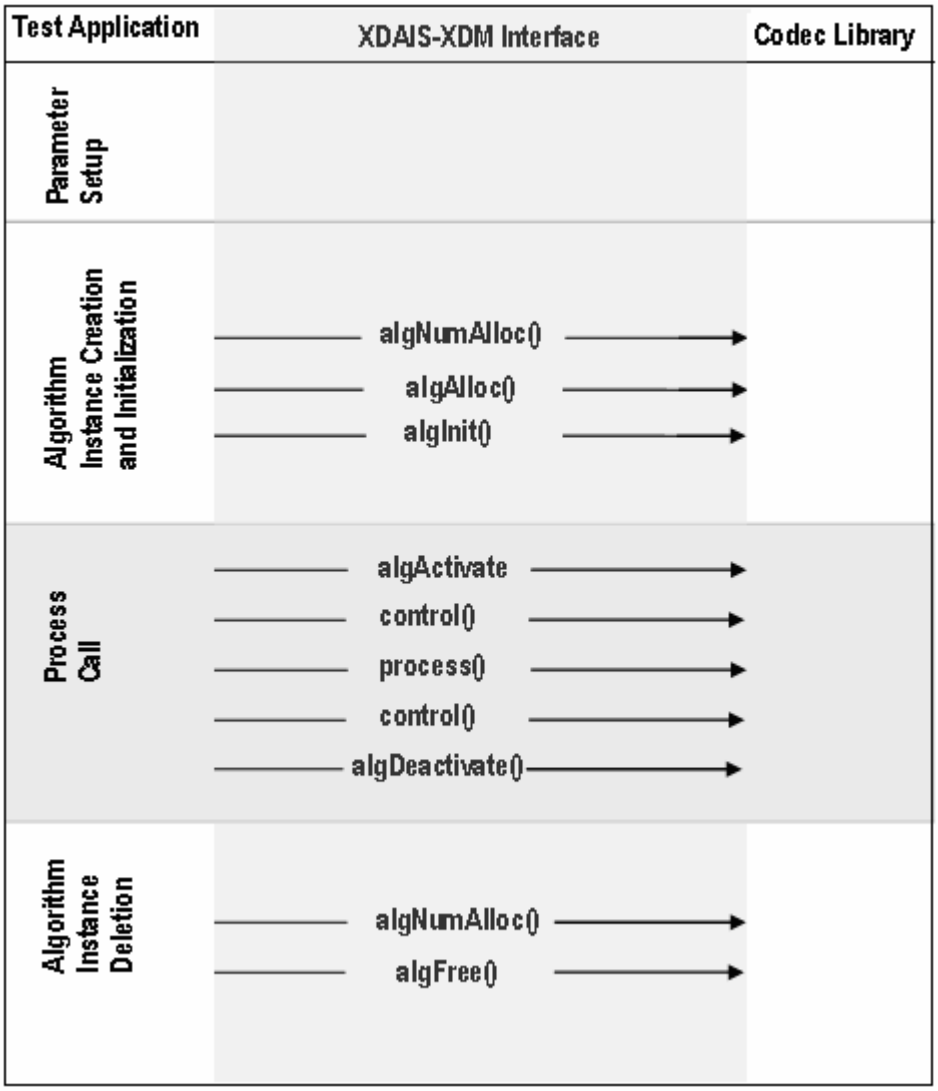


Figure 3-1. Test Application Sample Implementation

The test application is divided into four logical blocks:

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

3.1.1 Parameter Setup

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

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

1) Opens the generic configuration file, testvecs.cfg and reads the list of Encoder configuration file name (testparams.cfg).

2) Opens the Encoder configuration file, (testparams.cfg) and reads the various configuration parameters required for the algorithm.

   For more details on the configuration files, see Section 2.4.

3) Sets the `IVIDENC1_Params` structure based on the values it reads from the Testparams.cfg file.

4) Sets the extended parameters of the `IH264VENC_Params` structure based on the values it reads from the testparams.cfg file.

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

3.1.2 Algorithm Instance Creation and Initialization

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

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

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

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

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

After successful creation of the algorithm instance, the test application does DMA resource allocation for the algorithm.

Note:

DMAN3 function and IDMA3 interface is not implemented in DM365/DM368 codecs. Instead, it uses a DMA resource header file, which gives the framework the flexibility to change DMA resource to codec.

3.1.3 Process Call

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

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

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

3) Implements the process call based on the mode of operation – blocking or non-blocking. These different modes of operation are explained below. The behavior of the algorithm can be controlled using various dynamic parameters (see section 4.2.1.10). The inputs to the `process()` functions are input and output buffer descriptors, pointer to the `IVIDENC1_InArgs` and `IVIDENC1_OutArgs` structures.

4) Call the `process()` function to encode/decode a single frame of data. After triggering the start of the encode/decode frame start, the video task can be moved to SEM-pend state using semaphores. On receipt of interrupt signal for the end of frame encode/decode, the application should release the semaphore and resume the video task, which performs book-keeping operations and updates the output parameters structure -`IVIDENC1_OutArgs`.

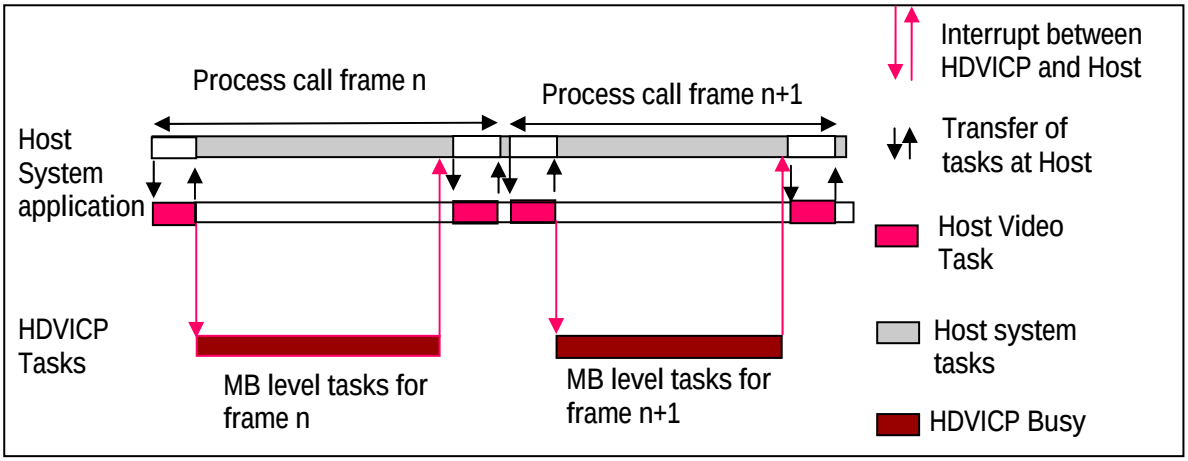


Figure 3-2. Process Call with Host Release

Note:

- The process call returns control to the application after the initial setup related tasks are completed.
- Application can schedule a different task to use the Host resource released free.
- All service requests from HDVICP are handled through interrupts.
- Application resumes the suspended process call after handling the last service request for HDVICP.
- Application can now complete concluding portions of the process call.

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

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

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

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

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

The for loop encapsulates frame level `process()` call and updates the input buffer and the output buffer pointer every time before the next call. The for loop runs for the designated number of frames and breaks-off whenever an error condition occurs.

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

3.1.4 Algorithm Instance Deletion

Once decoding/encoding is complete, the test application deletes the current algorithm instance The following APIs are called in a sequence:

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

2) `algFree()` - To query the algorithm to get the memory record information, which can be used by the application for freeing them up.

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

3.2 Handshaking Between Application and Algorithm

3.2.1 Resource Level Interaction

Following diagram explains the resource level interaction of the application with framework component and codecs. Application uses XDM for interacting with codecs. Similarly, it uses RMAN to grant resources to the codec.

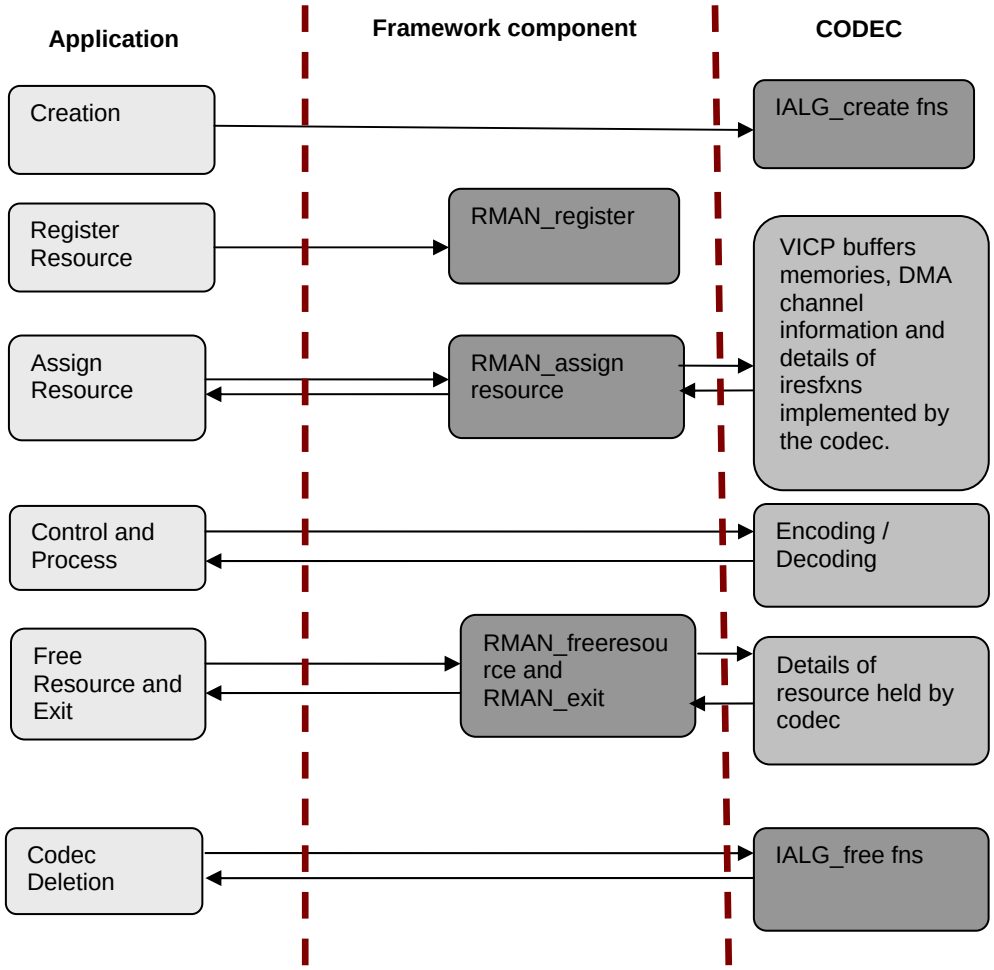


Figure 3-3. Resource Level Interaction.

3.2.2 Handshaking Between Application and Algorithms

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

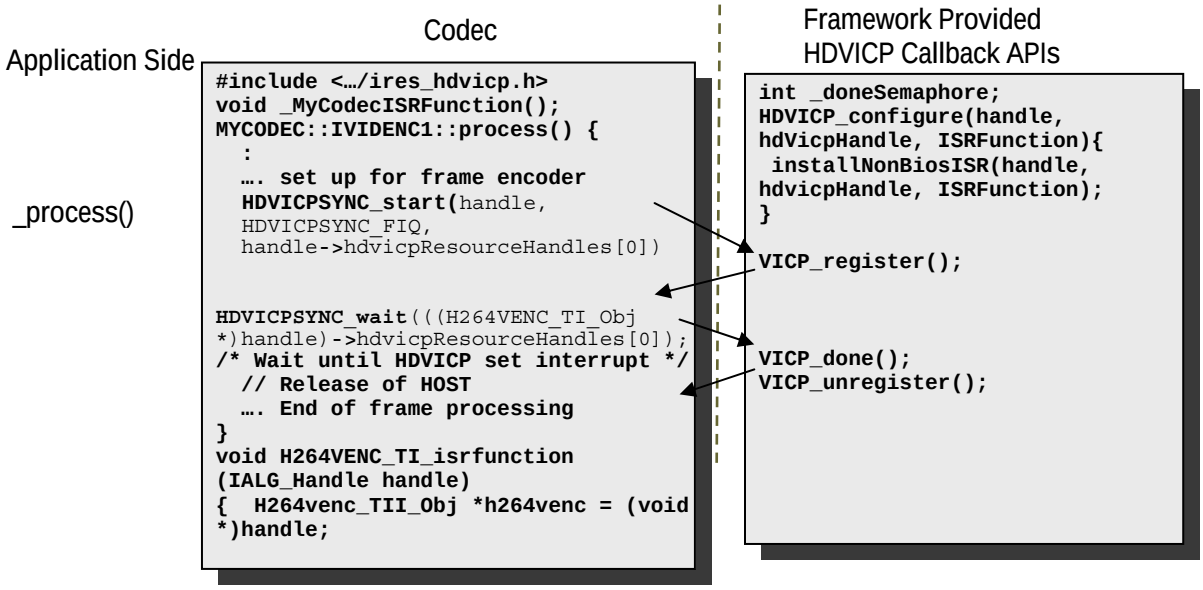


Figure 3-4. Interaction Between Application and Codec.

Note:

- Process call architecture shares Host resource among multiple threads.
- ISR ownership is with the FC resource manager – outside the codec.
- Codec implementation is OS independent.

The functions to be implemented by the application are:

1) `HDVICPSYNC_start(IALG_Handle handle, HDVICPSYNC_InterruptType intType, IRES_HDVICP_Handle hdvicpHandle)`

   This function is called by the algorithm to register the interrupt with the OS. This function also configures the Framework Component interrupt synchronization routine.

2) `HDVICPSYNC_wait (IRES_HDVICP_Handle hdvicpHandle)`

   This function is a FC call back function use to pend on a semaphore. Whenever the codec has completed the work on Host processor (after transfer of frame level encode/decode to HDVICP) and needs to relive the CPU for other tasks, it calls this function.

This function of FC implements a semaphore which goes into pend state and then the OS switches the task to another non-codec task.

Interrupts from HDVICP to Host ARM926 is used to inform when the frame processing is done. HDVICP sends interrupt which maps to `INT No 10 (KALINT9 Video MJCP)` of ARM926 INTC. After receiving this interrupt, the semaphore on which the codec task was waiting gets released and the execution resumes after the `HDVICPSYNC_wait()` function.

The following figure explains the interrupt interaction between application and codec.

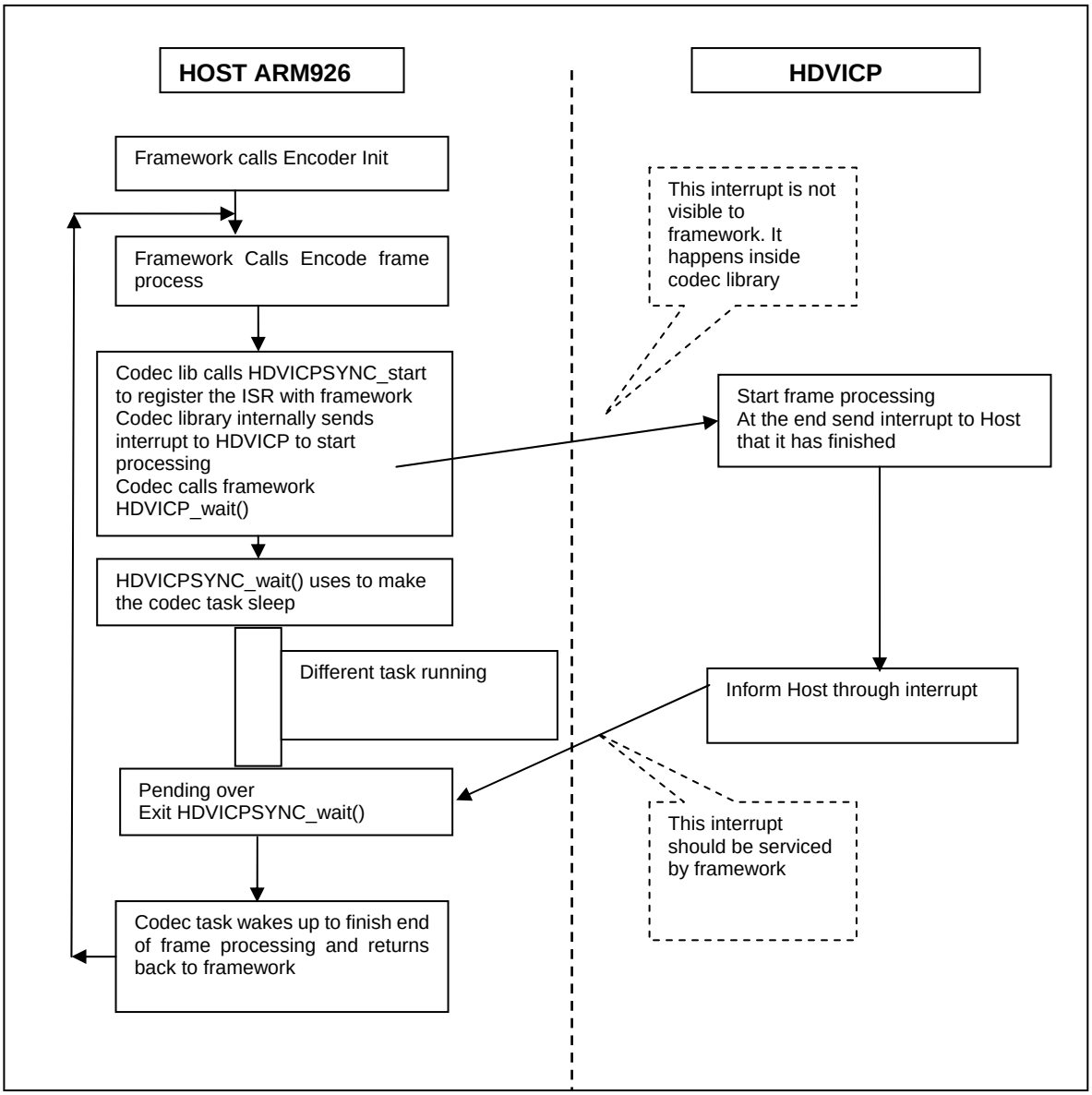


Figure 3-5. Interrupt Between Codec and Application.

3.3 Cache Management by Application

3.3.1 Cache Usage By Codec Algorithm

The codec source code and data, which runs on Host ARM926 can be placed in DDR. The host of DM365/DM368 has MMU and cache that the application can enable for better performance. Since the codec also uses DMA, there can be inherent cache coherency problems when application turns on the cache.

3.3.2 Cache and Memory Related Call Back Functions for Linux

To resolve the cache coherency and virtual to physical address issues, FC provides memory until library. These following functions can be used by codecs to resolve the cache coherency issues in Linux:

- `cacheInvalidate`
- `cacheWb`
- `cacheWbInv`
- `getPhysicalAddr`

3.3.2.1 cacheInvalidate

In cache invalidation process, the entries of the cache are deleted. This API invalidates a range of cache.

```
Void MEMUTILS_cacheInv (Ptr  addr, Int  sizeInBytes)
```

3.3.2.2 cacheWb

This API writes back cache to the cache source when it is necessary.

```
Void MEMUTILS_cacheWb (Ptr  addr, Int  sizeInBytes)
```

3.3.2.3 cacheWbInv

This API writes back cache to the cache source when it is necessary and deletes the cache contents.

```
Void MEMUTILS_cacheWbInv (Ptr  addr, Int  sizeInBytes)
```

3.3.2.4 getPhysicalAddr

This API obtains the physical address.

```
Void* MEMUTILS_getPhysicalAddr (Ptr  addr))
```

3.4 Sample Test Application

The test application exercises the `IVIDENC1` base class of the H.264 Encoder.

Table 3-1. process () Implementation

```
/* Main Function acting as a client for Video encode Call*/
/* Acquiring and intializing the resources needed to run the
encoder */
iresStatus = (IRES_Status) RMAN_init();
iresStatus = (IRES_Status) RMAN_register(&IRESMAN_EDMA3CHAN,
(IRESMAN_Params *)&configParams);

/*---------------- Encoder creation -----------------*/
handle = H264VENC_create(&fxns, &params)

/*Getting instance of algorithms that implements IALG and
IRES functions*/
iErrorFlag = RMAN_assignResources((IALG_Handle)handle,
                            &H264VENC_TI_IRES, /* IRES_Fxns* */
                              1 /* scratchId */);
/* Get Buffer information  */
iErrorFlag = H264VENC_control(
            handle,         // Instance Handle
            XDM_GETSTATUS,  // Command
            &dynamicparams, // Pointer to Dynamicparam structure
            &status         // Pointer to the status structure
        );
/*SET BASIC INPUT PARAMETERS */
iErrorFlag = H264VENC_control(
            handle,         // Instance Handle
            XDM_GETSTATUS,  // Command
            &dynamicparams, // Pointer to Dynamicparam structure
            &status         // Pointer to the status structure
        );
/* Based on the Num of buffers requested by the algorithm,
   the application will allocate for the same here
  */
AllocateH264IOBuffers(
            status, // status structure
            &inobj, // Pointer to Input Buffer Descriptor
&outobj) // Pointer to Output Buffer Descriptor
        );
/*Set Dynamic input parameters */
iErrorFlag = H264VENC_control(
            handle,         // Instance Handle
            XDM_GETSTATUS,  // Command
            &dynamicparams, // Pointer to Dynamicparam structure
            &status         // Pointer to the status structure
        );

/* for Loop for encode Call  for a given no of frames */
For(;;)
/* Read the input frame in the Application Input Buffer */
ReadInputData (inFile);
/*-----------------------------------------------------*/
/* Start the process : To start Encoding a frame      */
/* This will always follow a H264VENC_encode_end call */
```

```
/*-----------------------------------------------------*/

 iErrorFlag = H264VENC_encode (
                handle,   // Instance Handle    - Input
                &inobj,   // Input Buffers      - Input
                &outobj,  // Output Buffers     - Output
                &inargs,  // Input Parameters   - Input
                &outargs  // Output Parameters  - Output
            );
/* Get the statatus of the Encoder using control */
H264VENC_control(
                    handle,         // Instance Handle
                    XDM_GETSTATUS,  // Command - GET STATUS
                    &dynamicparams, // Input
                    &status         // Output
                );
     }
/* end of Do-While loop - which Encodes frames  */
/* Free Input and output buffers */
FreeH264IOBuffers(
            &inobj, // Pointer to Input Buffer Descriptor
&outobj // Pointer to Output Buffer Descriptor  );
/* Free assigned resources */
RMAN_freeResources((IALG_Handle)(handle),
                    &H264VENC_TI_IRES, /* IRES_Fxns* */
                   );
/* Delete the encoder Object handle*/
H264VENC_delete(handle);
/* Unregister protocal*/
RMAN_unregister(&IRESMAN_EDMA3CHAN);
RMAN_exit();
```

Note:

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

Chapter 4

API Reference

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

Topic **Page**

4.1 Symbolic Constants and Enumerated Data Types

This section 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.

4.1.1 Common XDM Symbolic Constants and Enumerated Data Types

Table 4-1. List of Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IVIDEO_FrameType	IVIDEO_I_FRAME	Intra coded frame
	IVIDEO_P_FRAME	Forward inter coded frame
	IVIDEO_B_FRAME	Bi-directional inter coded frame. Not supported in this version of H.264 Encoder.
	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. Not supported in this version of H.264 Encoder.
	IVIDEO_PI_FRAME	Interlaced frame, first field is a P frame, second field is an I frame. Not supported in this version of H.264 Encoder.
	IVIDEO_PP_FRAME	Interlaced frame, both fields are P frames.
	IVIDEO_PB_FRAME	Interlaced frame, first field is a P frame, second field is a B frame. Not supported in this version of H.264 Encoder.
	IVIDEO_BI_FRAME	Interlaced frame, first field is a B frame, second field is an I frame. Not supported in this version of H.264 Encoder.
	IVIDEO_BP_FRAME	Interlaced frame, first field is a B frame, second field is a P frame. Not supported in this version of H.264 Encoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_BB_FRAME	Interlaced frame, both fields are B frames. Not supported in this version of H.264 Encoder.
	IVIDEO_MBAFF_I_FRAME	Intra coded MBAFF frame. Not supported in this version of H.264 Encoder.
	IVIDEO_MBAFF_P_FRAME	Forward inter coded MBAFF frame. Not supported in this version of H.264 Encoder.
	IVIDEO_MBAFF_B_FRAME	Bi-directional inter coded MBAFF frame. Not supported in this version of H.264 Encoder.
	IVIDEO_MBAFF_IDR_FRAME	Intra coded MBAFF frame that can be used for refreshing video content. Not supported in this version of H.264 Encoder.
	IVIDEO_FRAMETYPE_DEFAULT	The default value is set to IVIDEO_I_FRAME.
IVIDEO_ContentType	IVIDEO_CONTENTTYPE_NA	Content type is not applicable. Encoder assumes IVIDEO_PROGRESSIVE.
	IVIDEO_PROGRESSIVE	Progressive video content. This is the default value.
	IVIDEO_INTERLACED	Interlaced video content.
IVIDEO_RateControlPreset	IVIDEO_NONE	No rate control is used
	IVIDEO_LOW_DELAY	Constant Bit-Rate (CBR) control for video conferencing.
	IVIDEO_STORAGE	Variable Bit-Rate (VBR) control for local storage and recording. This is the default value.
	IVIDEO_USER_DEFINED	User defined configuration using advanced parameters (extended parameters).
	IVIDEO_TWOPASS	Two pass rate control for non real time applications. Not supported in this version of H.264 Encoder.
	IVIDEO_RATECONTROLPRESET_DEFAULT	Set to IVIDEO_LOW_DELAY

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IVIDEO_SkipMode	IVIDEO_FRAME_ENCODED	Input content encoded
	IVIDEO_FRAME_SKIPPED	Input content skipped, that is, not encoded
	IVIDEO_SKIPMODE_DEFAULT	Default value is set to IVIDEO_FRAME_ENCODE
XDM_DataFormat	XDM_BYTE	Big endian stream. This is the default value.
	XDM_LE_16	16-bit little endian stream. Not supported in this version of H.264 Encoder.
	XDM_LE_32	32-bit little endian stream. Not supported in this version of H.264 Encoder.
XDM_ChromaFormat	XDM_CHROMA_NA	Chroma format not applicable. Encoder assumes IH264VENC_YUV_420IUV
	XDM_YUV_420P	YUV 4:2:0 planar. Not supported in this version of H.264 Encoder.
	XDM_YUV_422P	YUV 4:2:2 planar. Not supported in this version of H.264 Encoder.
	XDM_YUV_422IBE	YUV 4:2:2 interleaved (big endian). Not supported in this version of H.264 Encoder.
	XDM_YUV_422ILE	YUV 4:2:2 interleaved (little endian). Not supported in this version of H.264 Encoder.
	XDM_YUV_444P	YUV 4:4:4 planar. Not supported in this version of H.264 Encoder.
	XDM_YUV_411P	YUV 4:1:1 planar. Not supported in this version of H.264 Encoder.
	XDM_GRAY	Gray format. Not supported in this version of H.264 Encoder.
	XDM_RGB	RGB color format. Not supported in this version of H.264 Encoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_YUV_420SP	YUV 420 semiplanar (Luma 1st plane, * CbCr interleaved 2nd plane)
	XDM_ARGB8888	Alpha plane Not supported in this version of H.264 Encoder
	XDM_RGB555	RGB 555 color format Not supported in this version of H.264 Encoder
	XDM_RGB565	RGB 556 color format Not supported in this version of H.264 Encoder
	XDM_YUV_444ILE	YUV 4:4:4 interleaved (little endian) Not supported in this version of H.264 Encoder
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
	XDM_SETDEFAULT	Initialize all fields in DynamicParams structure to default values specified in the library
	XDM_FLUSH	Handle end of stream conditions. This command forces algorithm instance to output data without additional input. Not supported in this version of H.264 Encoder.
	XDM_GETVERSION	Query the algorithm version.
	XDM_GETBUFINFO	Query algorithm instance regarding the properties of input and output buffers.
XDM_EncodingPreset	XDM_DEFAULT	Default setting of the algorithm specific creation time parameters. This uses XDM_HIGH_QUALITY settings.
	XDM_HIGH_QUALITY	Set algorithm specific creation time parameters for high quality (default setting).

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_HIGH_SPEED	Set algorithm specific creation time parameters for high speed.
	XDM_USER_DEFINED	User defined configuration using advanced parameters.
XDM_EncMode	XDM_ENCODE_AU	Encode entire access unit. This is the default value.
	XDM_GENERATE_HEADER	Encode only header.
XDM_ErrorBit	XDM_APPLIEDCONCEALMENT	Bit 9 ❑ 1 – Applied concealment ❑ 0 – Ignore
	XDM_INSUFFICIENTDATA	Bit 10 ❑ 1 – Insufficient data ❑ 0 – Ignore
	XDM_CORRUPTEDDATA	Bit 11 ❑ 1 – Data problem/corruption ❑ 0 – Ignore
	XDM_CORRUPTEDHEADER	Bit 12 ❑ 1 – Header problem/corruption ❑ 0 – Ignore
	XDM_UNSUPPORTEDINPUT	Bit 13 ❑ 1 – Unsupported feature/parameter in input ❑ 0 – Ignore
	XDM_UNSUPPORTEDPARAM	Bit 14 ❑ 1 – Unsupported input parameter or configuration ❑ 0 – Ignore
	XDM_FATALERROR	Bit 15 ❑ 1 – Fatal error (stop encoding) ❑ 0 – Recoverable error

Note:

- encodingPreset: There are no tools which can cause perfromance difference. Hence, XDM_HIGH_QUALITY and XDM_HIGH_SPEED will give the same bitstream/perfromance.
- The remaining bits that are not mentioned in XDM_ErrorBit are interpreted as:
  - Bit 16-32: Reserved
  - Bit 8: Reserved

- Bit 0-7: Codec and implementation specific

The algorithm can set multiple bits to 1 depending on the error condition.

4.1.2 H264 Encoder Symbolic Constants and Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IH264VENC_Level	IH264VENC_LEVEL_10	Level 1.0 identifier for H.264 Encoder
	IH264VENC_LEVEL_1b	Level 1.b identifier for H.264 Encoder
	IH264VENC_LEVEL_11	Level 1.1 identifier for H.264 Encoder
	IH264VENC_LEVEL_12	Level 1.2 identifier for H.264 Encoder
	IH264VENC_LEVEL_13	Level 1.3 identifier for H.264 Encoder
	IH264VENC_LEVEL_20	Level 2.0 identifier for H.264 Encoder
	IH264VENC_LEVEL_21	Level 2.1 identifier for H.264 Encoder
	IH264VENC_LEVEL_22	Level 2.2 identifier for H.264 Encoder
	IH264VENC_LEVEL_30	Level 3.0 identifier for H.264 Encoder
	IH264VENC_LEVEL_31	Level 3.1 identifier for H.264 Encoder
	IH264VENC_LEVEL_32	Level 3.2 identifier for H.264 Encoder
	IH264VENC_LEVEL_40	Level 4.0 identifier for H.264 Encoder
	IH264VENC_LEVEL_41	Level 4.1 identifier for H.264 Encoder
	IH264VENC_LEVEL_42	Level 4.2 identifier for H.264 Encoder
	IH264VENC_LEVEL_50	Level 5.0 identifier for H.264 Encoder

4.1.3 H264 Encoder Error code Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IH264VENC_STATUS	IH264VENC_ERR_MAXWIDTH	maxWidth not supported. “Fatal input error” is returned in algInit instance creation stage if maxWidth in the input params exceeds H264VENC_TI_MAX_WIDTH (2048) or is less than H264VENC_TI_MIN_WIDTH H264VENC_TI_MIN_WIDTH takes value of 128 in case of encodingPreset = XDM_USER_DEFINED and encQuality = 0 OR 320 in case of encodingPreset = XDM_HIGH_SPEED/XDM_HIGH_QUALITY.
	IH264VENC_ERR_MAXHEIGHT	maxHeight not supported. fatal input error is returned in algInit instance creation stage if maxHeight in input params exceeds H264VENC_TI_MAX_HEIGHT (2048) or is less than H264VENC_TI_MIN_HEIGHT H264VENC_TI_MIN_HEIGHT takes value of 96 in case of encodingPreset = XDM_USER_DEFINED and encQuality = 0 OR 128 in case of encodingPreset = XDM_HIGH_SPEED/XDM_HIGH_QUALITY.
	IH264VENC_ERR_ENCODINGPRESET	encodingPreset not supported fatal input error” is returned during algInit if the encodingPreset parameter is out of supported range XDM_DEFAULT (0) to XDM_USER_DEFINED (3) inclusive .
	IH264VENC_ERR_RATECONTROLPRESET	rateControlPreset not supported fatal input error is returned during algInit if the rateControlPreset parameter is out of supported range 0 to IVIDEO_USER_DEFINED (5) inclusive.
	IH264VENC_ERR_MAXFRAMERATE	maxFrameRate not supported. “Fatal input error” is returned during algInit if maxFrameRate exceeds max supported value of 120000 or is less than 0.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_MAXBITRATE	maxBitRate not supported fatal input error is returned during algInit if maxBitRate exceed max supported value of 50000000 or is less than 0.
	IH264VENC_ERR_DATAENDIANNESS	dataEndianness not supported fatal input error is returned during algInit if dataEndianness is not set to XDM_BYTE.
	IH264VENC_ERR_INPUTCHROMAFORMAT	inputChromaFormat not supported fatal input error is returned during algInit if inputChromaFormat is not set to XDM_YUV_420SP or XDM_CHROMA_NA.
	IH264VENC_ERR_INPUTCONTENTTYPE	inputContentType not supported fatal input error is returned during algInit if inputContentType is not set to IVIDEO_PROGRESSIVE or IVIDEO_INTERLACED. This error is also returned during algInit if interlaced encoding is enabled (inputContentType set to IVIDEO_INTERLACED) for levels less than 2.1 or more than 4.1.
	IH264VENC_ERR_RECONCHROMAFORMAT	reconChromaFormat not supported fatal input error is returned during algInit if reconChromaFormat is not set to XDM_YUV_420SP or XDM_CHROMA_NA.
	IH264VENC_ERR_INPUTWIDTH	inputWidth not supported fatal input error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if the inputWidth in input dynamic params exceeds maxWidth or if inputWidth is less than H264VENC_TI_MIN_WIDTH or not multiple of 2. Control call returns IVIDENC1_EFAIL. H264VENC_TI_MIN_WIDTH takes value of 128 in case of encodingPreset = XDM_USER_DEFINED and encQuality = 0 OR 320 in case of encodingPreset = XDM_HIGH_SPEED/XDM_HIGH_QUALITY.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_INPUTHEIGHT	inputHeight not supported fatal input error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if the inputHeight in input dynamic params exceeds maxHeight or if inputHeight is less than H264VENC_TI_MIN_HEIGHT or not multiple of 2 for progressive content and not multiple of 4 for interlaced content. Control call returns IVIDENC1_EFAIL. H264VENC_TI_MIN_HEIGHT takes value of 96 in case of encodingPreset = XDM_USER_DEFINED and encQuality = 0 OR 128 in case of encodingPreset = XDM_HIGH_SPEED/XDM_HIGH_QUALITY.
	IH264VENC_ERR_MAX_MBS_IN_FRM_LIMIT_EXCEED	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if the number of MBs in a frame exceeds the maximum limit for resolution of 2048x2048. Control call returns IVIDENC1_EFAIL
	IH264VENC_ERR_TARGETFRAMERATE	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if targetFrameRate in dynamic params exceeds maxFrameRate or is less than 0 or not a multiple of 500. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_TARGETBITRATE	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if targetBitRate in dynamic params exceeds maxBitRate or less than 0. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_PROFILEIDC	profileIdc not supported fatal input error is returned during algInit if profileIdc is not 66 (BP) or 77(MP) or 100 (H)

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_LEVELIDC	levelIdc not supported fatal input error is returned during g algInit if levelIdc is not as per IH264VENC_Level range IH264VENC_LEVEL_1b(9) to IH264VENC_LEVEL_50(50)
	IH264VENC_ERR_ENTROPYMODE_IN_BP	entropyMode not supported, a fatal input error is returned during algInit if entropyMode is 1 (CABAC) for Baseline Profile (profileIdc = 66) or if the value is out of the supported range 0 or 1 for Main/High Profile (profileIdc = 77/100)
	IH264VENC_ERR_TRANSFORM8X8FLAGINTRA_IN_BP_MP	transform8x8FlagIntraFrame not supported, a fatal input error is returned during algInit if transform8x8FlagIntraFrame is enabled for Baseline or Main Profile (profileIdc = 66 or 77) or if the value is out of the supported range 0 or 1 for High Profile (profileIdc = 100)
	IH264VENC_ERR_TRANSFORM8X8FLAGINTER_IN_BP_MP	transform8x8FlagInterFrame not supported, a fatal input error is returned during algInit if transform8x8FlagInterFrame is enabled for Baseline or Main Profile (profileIdc = 66 or 77) or if the value is out of the supported range 0 or 1 for High Profile (profileIdc = 100)
	IH264VENC_ERR_SEQSCALINGFLAG_IN_BP_MP	seqScalingFlag not supported, a fatal input error is returned during algInit if seqScalingFlag is enabled for Baseline or Main Profile (profileIdc = 66 or 77) or if the value is out of the supported range from 0:4 for High Profile (profileIdc = 100)
	IH264VENC_ERR_ASPECTRATIOX	aspectRatioX not supported fatal input error is returned during algInit if aspectRatioX is lesst than 1.
	IH264VENC_ERR_ASPECTRATIOY	aspectRatioY not supported fatal input error is returned during algInit if aspectRatioY is less than 1.
	IH264VENC_ERR_PIXELRANGE	pixelRange n not supported fatal input error is returned during algInit if pixelRange is not 0 or 1.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_TIMESCALE	timeScale not supported fatal input error is returned during algInit if timeScale is less than 0 or if timeScale * 1000 exceeds targetFrameRate.
	IH264VENC_ERR_NUMUNITSINTICKS	numUnitsInTicks not supported fatal input error is returned during algInit if numUnitsInTicks is less than 0.
	IH264VENC_ERR_ENABLEVUIPARAMS	enableVUIparams not supported fatal input error is returned during algInit if enableVUIparams is not 0, 1 or 2.
	IH264VENC_ERR_RESETHDVICPEVERYFRAME	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if resetHDVICPeveryFrame extended dynamic parameter is not 0 or 1. Control call returns IVIDENC1_EFAIL..
	IH264VENC_ERR_MEALGO	meAlgo not supported fatal input error is returned during algInit if meAlgo is not 0 or 1.
	IH264VENC_ERR_UNRESTRICTEDMV	unrestrictedMV not supported fatal input error is returned during algInit if unrestrictedMV is not 0 or 1.
	IH264VENC_ERR_ENCQUALITY	encQuality not supported fatal input error is returned during algInit if encQuality is not 0, 1 or 2.
	IH264VENC_ERR_ENABLEARM926TCM	enableARM926Tcm not supported fatal input error is returned during algInit if enableARM926Tcm is not 0 or 1. This error is also returned if enableARM926Tcm is 1 for maxWidth greater than 1280.
	IH264VENC_ERR_ENABLEDDRBUFF	mapIMCOPtoDDR not supported fatal input error is returned during algInit if mapIMCOPtoDDR is not 0 or 1.
	IH264VENC_ERR_SLICEMODE	sliceMode not supported fatal input error is returned during algInit if sliceMode is not 0, 1, 2 or 3
	IH264VENC_ERR_OUTPUTDATAMODE	oututDataMode not supported fatal input error is returned during algInit if outputDataMode is not 0 or 1.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_SLICEFORMAT	sliceFormat not supported fatal input error is returned during algInit if sliceFormat is not 0 or 1.
	IH264VENC_ERR_LEVEL_NOT_FOUND	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if inputWidth, inputHeight, targetBitRate and targetFrameRate are not compliant to Level limits specified in *Table A-1 of ISO/IEC 14496-10*. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_REFFRAMERATE_MISMATCH	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if refFrameRate and targetFrameRate mismatch. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_INTRAFRAMEINTERVAL	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if intraFrameInterval is less than 0. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_GENERATEHEADER	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if generateHeader is not 0 or 1. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_FORCEFRAME	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if forceFrame is not IVIDEO_NA_FRAME or IVIDEO_I_FRAME or IVIDEO_IDR_FRAMEE. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_RCALGO	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if rcAlgo is not set to 0, 1 or 2 when rcPreset is IVIDEO_USER_DEFINED. Control call returns IVIDENC1_EFAIL.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_INTRAFRAMEQP	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if intraFrameQP is less than 0 or more than 51. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_INTERPFRAMEQP	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if interPFrameQP is less than 0 or more than 51. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_RCQMAX	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if rcQMax is less than 0 or more than 51. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_RCQMIN	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if rcQMin is less than 0 or more than rcQMax. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_RCIQMAX	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if rcQMaxI is less than 0 or more than 51. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_RCIQMIN	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if rcQMinI is less than 0 or more than rcQMaxI. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_INITQ	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if initQ is less than -1 or more than 51. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_MAXDELAY	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if maxDelay exceeds 10000. Control call returns IVIDENC1_EFAIL.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_LFDISABLEIDC	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if lfDisableIdc is less than 0 or more than 2. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_ENABLEBUFSEI	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if enableBufSEI is not 0 or 1. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_ENABLEPICTIMSEI	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if enablePicTimSEI is not 0 or 1. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_SLICESIZE	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if sliceSize is not with in range. The range depends on the value of sliceMode. Control call returns IVIDENC1_EFAIL. See the note at end of section 4.2.2.2 for more details on range and interpretation of sliceSize.
	IH264VENC_ERR_INTRASLICENUM	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if intraSliceNum is less than 0. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_AIRRATE	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if airRate is less than of 0. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_MEMULTIPART	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if meMultiPart is not 0 or 1. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_INTRATHRQF	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if intraThrQF is less than 0 or more than 5. Control call returns IVIDENC1_EFAIL.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_PERCEPTUALRC	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if perceptualRC is not 0 or 1. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_IDRFRAMEINTERVAL	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if idrFrameInterval is less than 0. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_MVSADOUTFLAG	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if mvSADoutFlag is not 0 or 1. Control call returns IVIDENC1_EFAIL
	IH264VENC_ERR_ENABLEROI	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if enableROI is not 0 or 1. Control call returns IVIDENC1_EFAIL
	IH264VENC_ERR_METADATAFLAG	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if metaDataGenerateConsume is not set between 0 and 3. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_MAXINTERFRAMEINTERVAL	This fatal unsupported param error is returned in algInit instance creation if maxInterFrameInterval not 0 or 1.
	IH264VENC_ERR_CAPTUREWIDTH	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if captureWidth is not 0 and less than inputWidth. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_INTERFRAMEINTERVAL	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if interFrameInterval is not 0 or 1. Control call returns IVIDENC1_EFAIL..
	IH264VENC_ERR_MBDATAFLAG	This warning is returned in videncStatus.extendedError during XDM_SETPARAMS control call if mbDataFlag is not set to 0. Control call returns IVIDENC1_EFAIL.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_IVIDENC1_DYNAMICPARAMS_SIZE_IN_CORRECT	This fatal error is returned in videncStatus.extendedError during a control call if dynamic param size is not IVIDENC1_DynamicParams or IH264VENC_DynamicParams. Control call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_IVIDENC1_PRO CESS_ARGS_NULL	This fatal error is returned in process call if any of input handle or inBufs or inArgs or outBufs are NULL.
	IH264VENC_ERR_IVIDENC1_INARGS_SIZE	This fatal error is returned in outArgs->extendedError if inargs size in process call is not set to IVIDENC1_InArgs or IH264VENC_InArgs. This error is returned provided OutArgs size is set correctly. Process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_IVIDENC1_OUTARGS_SIZE	This fatal error can be retrieved from a XDM_GETSTATUS control call if outArgs size in process call was not set to IVIDENC1_OutArgs or IH264VENC_OutArgs. Process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_IVIDENC1_INARGS_INPUTID	This fatal error is returned in outArgs->extendedError if inputID in inArgs of process call is 0. Process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_IVIDENC1_INARGS_TOPFIELDFIRSTFLAG	This fatal error is retruned in outArgs->extendedError if topFieldFirstFlag in inArgs is not set correctly to 0 or 1 for interlace content. Process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_IVIDENC1_INBUFS	This fatal error is returned in outArgs->extendedError if inBufs is null or if numBufs in inBufs is not set to 2 or if frameWidth and frameHeight in inBufs are not equal to inputWidth and inputHeight of XDM_SETPARAMS control call. Process call returns IVIDENC1_EFAIL.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_IVIDENC1_INBUFS_BUFDESC	This fatal error is returned in outArgs->extendedError if buffer descriptors in inBufs are either NULL or if their sizes are less than the frame size. Process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_IVIDENC1_OUTBUFS	This fatal error is returned in outArgs->extendedError if outBufs is NULL or if numBufs in outBufs is less than 1, Process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_IVIDENC1_OUTBUFS_NULL	This fatal error is returned in outArgs->extendedError if bufs or bufSizes of outBufs is NULL, Process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_IVIDENC1_INVALID_NUM_OUTDATA_UNIT	This fatal error is returned in outArgs->extendedError if numOutputDataUnits is not valid. Valid values are 1 to IH264VENC_TI_MAXNUMBLOCKS. Process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_INTERLACE_IN_BP	This fatal error is returned during algInit() instance creation stage if application tries to encode interlaced content in Baseline Profile mode.
	IH264VENC_ERR_INSERTUSERDATA	This fatal error is returned in outArgs->extendedError if insertUserData in extended inArgs is not 0 or 1, Process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_LENGTHUSERDATA	This fatal error is returned in outArgs->extendedError if lengthUserData in extended inArgs is less than 0. Process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_ROIPARAM	This fatal error is returned in outArgs->extendedError if ROI parameters in extended inArgs are not set correctly. Process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_PROCESS_CALL	This fatal error is returned in outArgs->extendedError if process call encounters a fatal error during execution. Process call returns IVIDENC1_EFAIL.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_HANDLE_NULL	This fatal error is returned when input handle is NULL. If the handle is NULL in algFree or algInit call this error is returned to call function. If the handle is NULL in a control call this error is returned in sStatus->videncStatus.extendedError and control call returns IVIDENC1_EFAIL. If the handle is NULL in a process call this error is returned in outArgs->extendedError and process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_INCORRECT_HANDLE	This fatal error is returned when incorrect codec handle is passed to code API. If the handle is incorrectly passed in algFree or algInit call this error is returned to callee function. If the handle is incorrectly passed in a control call this error is returned in sStatus->videncStatus.extendedError and control call returns IVIDENC1_EFAIL. If the handle is incorrectly passed in a process call this error is returned in outArgs->extendedError and process call returns IVIDENC1_EFAIL.
	IH264VENC_ERR_MEMTAB_NULL	This fatal error is returned when memtabs passed to algInit or algFree are NULL or not aligned to 32bit word boundary.
	IH264VENC_ERR_IVIDENC1_INITPARAMS_SIZE	This fatal error is returned when size of algParams passed to algInit is not set to size of IVIDENC1_Params or size of IH264VENC_Params.
	IH264VENC_ERR_MEMTABS_BASE_NULL	This fatal error is returned when base pointer of memTabs passed to algInit are NULL.
	IH264VENC_ERR_MEMTABS_BASE_NOT_ALIGNED	This fatal error is returned when base pointer of memTabs passed to algInit are not aligned as per the requested alignment specified in algAlloc.
	IH264VENC_ERR_MEMTABS_SIZE	This fatal error is returned when size of memTabs passed to algInit are less than the requested size specified in algAlloc.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_MEMTABS_ATTRS	This fatal error is returned when attrs of memTabs passed to algInit are not as per the requested attrs specified in algAlloc.
	IH264VENC_ERR_MEMTABS_SPACE	This fatal error is returned when space of memTabs passed to algInit are not as per the requested space specified in algAlloc.
	IH264VENC_ERR_MEMTABS_OVERLAP	This fatal error is returned any two memTabs passed to algInit are overlapping in memory.
	IH264VENC_ERR_CODEC_INACTIVE	This fatal error is returned when codec process call or control call is made without activating it. If a control call is made without aprior algActivate this error is returned in sStatus->videncStatus.extendedError and control call returns IVIDENC1_EFAIL. If a process call is made without aprior algActivate this error is returned in outArgs->extendedError and process call returns IVIDENC1_EFAIL
	IH264VENC_WARN_LEVELIDC	This warning is returned in videncStatus.extendedError in XDM_GETSTATUS control call after instance creation if leveldc during instance creation is not set to valid level enumerations range from IH264VENC_LEVEL_1b to IH264VENC_LEVEL_50. Encoder would continue assuming levelIdc as IH264VENC_LEVEL_50.
	IH264VENC_WARN_RATECONTROLPRESET	This warning is returned in videncStatus.extendedError in XDM_GETSTATUS control call after instance creation if rcPreset is neither of IVIDEO_NONE or IVIDEO_LOW_DELAY or IVIDEO_STORAGE. Encoder would continue by assuming rcPreset is IVIDEO_LOW_DELAY.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IH264VENC_ERR_STATUS_BUF	This warning is returned in videncStatus.extendedError during XDM_GETVERSION control call if videncStatus.data.buf is NULL or if videncStatus.data.bufSize is insufficient to copy the library version string. The control call returns IVIDENC1_EFAIL.

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:

- `XDM_BufDesc`
- `XDM1_BufDesc`
- `XDM_SingleBufDesc`
- `XDM1_SingleBufDesc`
- `XDM_AlgBufInfo`
- `IVIDEO_BufDesc`
- `IVIDEO1_BufDescIn`
- `IVIDENC1_Fxns`
- `IVIDENC1_Params`
- `IVIDENC1_DynamicParams`
- `IVIDENC1_InArgs`
- `IVIDENC1_Status`
- `IVIDENC1_OutArgs`

4.2.1.1 XDM_BufDesc

Description

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

Fields

Field	Data type	Input/ Output	Description
`**bufs`	`XDAS_Int8`	Input	Pointer to the vector containing buffer addresses
`numBufs`	`XDAS_Int32`	Input	Number of buffers
`*bufSizes`	`XDAS_Int32`	Input	Size of each buffer in bytes

4.2.1.2 XDM_AlgBufInfo

Description

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

Fields

Field	Data type	Input/ Output	Description
`minNumInBufs`	`XDAS_Int32`	Output	Number of input buffers
`minNumOutBufs`	`XDAS_Int32`	Output	Number of output buffers
`minInBufSize[XDM_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	Size in bytes required for each input buffer
`minOutBufSize[XDM_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	Size in bytes required for each output buffer

Note:

For H.264 Base/Main/High Profile Encoder, the buffer details are:

- Number of input buffer required is 2 for YUV 420P with chroma interleaved.
- Number of output buffer required is 1.
- The input buffer sizes (in bytes) for worst case 2048x2048 format are:

  For YUV 420P:
  Y buffer = 2048 * 2048
  UV buffer = 2048 * 1024

The above input buffer size calculation is done assuming that the

capture width is same as input width. For details on capture width, see Section 4.2.1.10.

For interlaced sequence, encoder ignores the input field buffers if they are stored in interleaved or non-interleaved format. But, it expects the start pointer of top or bottom field be given to it during the process call of the top or bottom fields, respectively. The pitch to move to the next line of the field is conveyed using `captureWidth` of `DynamicParams`.

- There is no restriction on output buffer size except that it should be enough to store one frame of encoded data.The output buffer size returned by the `XDM_GETBUFINFO` command assumes that the worst case output buffer size is (`frameHeight*frameWidth`)/2.
- In case of STP, low resolution needs an extra output buffer to pass metadata information from codec to application. High resolution needs an extra input buffer to pass metadata information from application to codec. The metadata is copied from output buffer of low resolution encoder to the input buffer of high resolution encoder.

These are the maximum buffer sizes, but you can reconfigure depending on the format of the bit-stream.

4.2.1.3 XDM1_BufDesc

Description

This structure defines the buffer descriptor for input and output buffers in XDM 1.0 IVIDENC1.

Fields

Field	Data type	Input/ Output	Description
`numBufs`	`XDAS_Int32`	Input	Number of buffers
`descs[XDM_MAX_IO_BUFFERS]`	`XDM1_SingleBufDesc`	Input	Array of buffer descriptors.

4.2.1.4 XDM_SingleBufDesc

Description

This structure defines the single buffer descriptor for input and output buffers in XDM 1.0 IVIDENC1.

Fields

Field	Data type	Input/ Output	Description
`*buf`	`XDAS_Int8`	Input	Pointer to a buffer address
`bufSize`	`XDAS_Int32`	Input	Size of the buffer in bytes

4.2.1.5 XDM1_SingleBufDesc

Description

This structure defines the single buffer descriptor for input and output buffers in XDM 1.0 IVIDENC1.

Fields

Field	Data type	Input/ Output	Description
*buf	XDAS_Int8	Input	Pointer to a buffer address
bufSize	XDAS_Int32	Input	Size of buffer in bytes
accessMask	XDAS_Int32	Input	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 CPU), then bits in this mask should not be set. **Note**: This feature is not supported in this version of H264 Encoder.

4.2.1.6 IVIDEO_BufDesc

Description

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

Fields

Field	Data type	Input/ Output	Description
numBufs	XDAS_Int32	Input	Number of buffers
width	XDAS_Int32	Input	Padded width of the video data
*bufs[XDM_MAX_IO_BUFFERS]	XDAS_Int8	Input	Pointer to the vector containing buffer addresses
bufSizes[XDM_MAX_IO_BUFFERS]	XDAS_Int32	Input	Size of each buffer in bytes
numBufs	XDAS_Int32	Input	Number of buffers

4.2.1.7 IVIDEO1_BufDescIn

Description

This structure defines the buffer descriptor for input 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. **Note**: It will be same as inputWidth for width multiple of 16. For inputWidth non-multiple of 16, application will set this field to next multiple of 16.
frameHeight	XDAS_Int32	Input	Height of the video frame. **Note**: **Progressive**: It will be same as inputHeight for height multiple of 16.For inputHeight non-multiple of 16, application will set this field to next multiple of 16. **Interlaced**: It will be same as inputHeight for height multiple of 32.For inputHeight non-multiple of 32, application will set this field to next multiple of 32.
framePitch	XDAS_Int32	Input	Frame pitch used to store the frame. This field is not used by the encoder.
bufDesc[XDM_MAX_IO_BUFFERS]	XDM1_SingleBufDesc	Input	Picture buffers

4.2.1.8 IVIDENC1_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.

Field	Data type	Input/ Output	Description
			For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).
*process	XDAS_Int32	Input	Pointer to the process() function.
*control	XDAS_Int32	Input	Pointer to the control() function.

4.2.1.9 IVIDENC1_Params

Description

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

Fields

Field	Data type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes. Default size is size of IH264VENC_PARAMS structure.
encodingPreset	XDAS_Int32	Input	Encoding preset. See XDM_EncodingPreset enumeration for details.. Default value = XDM_USER_DEFINED.
rateControlPreset	XDAS_Int32	Input	Rate control preset. See IVIDEO_RateControlPreset enumeration for details. Default value = IVIDEO_STORAGE.
maxHeight	XDAS_Int32	Input	Maximum video height to be supported in pixels. Default value = 1088
maxWidth	XDAS_Int32	Input	Maximum video width to be supported in pixels. Default value = 1920.
maxFrameRate	XDAS_Int32	Input	Maximum frame rate in fps * 1000 to be supported. Default value = 120000.
maxBitRate	XDAS_Int32	Input	Maximum bit-rate to be supported in bits per second. Default value = 50000000.

Field	Data type	Input/ Output	Description
dataEndianness	XDAS_Int32	Input	Endianness of input data. See XDM_DataFormat enumeration for details. Default value = XDM_BYTE.
maxInterFrameInterval	XDAS_Int32	Input	Distance from I-frame to P-frame: ❑ 1 - If no B-frames ❑ 2 - To insert one B-frame This parameter is not supported as B-frames are not supported. Set value = 1
inputChromaFormat	XDAS_Int32	Input	Input chroma format. See XDM_ChromaFormat and IH264VENC_ChromaFormat enumeration for details. Set value as = XDM_YUV_420SP. Other values are not supported.
inputContentType	XDAS_Int32	Input	Input content type. See IVIDEO_ContentType enumeration for details. Default value = IVIDEO_PROGRESSIVE.
reconChromaFormat	XDAS_Int32	Input	Chroma formats for the reconstruction buffers. Set value as = XDM_YUV_420SP. Other values are not supported.

Note:

encodingPreset: There are no tools which can cause perfromance difference. Hence, XDM_HIGH_QUALITY and XDM_HIGH_SPEED will give the same bitstream/perfromance.

The maximum video height and width supported are 2048 and 2048 pixels respectively.

For the supported maxBitRate values, see Annex A in *ISO/IEC 14496-10*.

The following fields of IVIDENC1_Params data structure are level dependent:

- maxHeight
- maxWidth
- maxFrameRate
- maxBitRate

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.

The maximum value for `maxFrameRate` and `maxBitRate` is 120 (120000) and 50000000 respectively.

Use the following expression to check the supported `maxFrameRate` values for each level:

`maxFrameRate` <= `maxMbsPerSecond` / `FrameSizeinMbs`;

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`;

See Table A.1 – Level Limits in *ISO/IEC 14496-10* for the supported values of Max Video bit-rate.

During creation time, these values are checked against the maximum values defined for the encoder. If the specified values exceed or do not match the limit supported by encoder, the encoder continues to encode with the next higher supported level. Since the actual height and width are specified later using control operation with dynamic parameters, the level based checking is done during the control operation.

4.2.1.10 IVIDENC1_DynamicParams

Description

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

Fields

Field	Data type	Input/ Output	Description
`size`	`XDAS_Int32`	Input	Size of the basic or extended (if being used) data structure in bytes. Default value is size of `IVIDENC1_DynamicParams` structure.
`inputHeight`	`XDAS_Int32`	Input	Height of input frame in pixels. Input height can be changed before start of encoding within the limits of maximum height set in creation phase. `inputHeight` must be multiple of two. Minimum height supported is 128. Irrespective of interlaced or progressive content, input height should be given as frame height. For any height lesser than 128 and greater than 96, you can use version 1.1, backward compatible mode See section 1.5 for details **Note**: **Progressive**: When the input height is a non-multiple of 16, the encoder expects the application to pad the input frame to the nearest multiple of 16 at the bottom of the frame. In this case, the application should set input height to actual height but should provide the padded input YUV data buffer to encoder. The encoder then sets the difference of the actual height and padded height as crop information in the bit-stream. **Interlaced**: When the input height is a non-multiple of 32, the encoder expects the application to pad the input frame to the nearest multiple of 32 at the bottom of the frame. In this case, the application should set input height to actual height but should provide the padded input YUV data buffer to encoder. The encoder then sets the difference of the actual height and padded height as crop information in the bit-stream. Default value = 576.

Field	Data type	Input/ Output	Description
inputWidth	XDAS_Int32	Input	Width of input frame in pixels. Input width can be changed before the start of encoding within the limits of maximum width set in creation phase. inputWidth must be multiples of two. Minimum width supported by encoder is 320. For any width lesser than 320 and greater than 128, you can use version 1.1, backward compatible mode. See section 1.5 for details. **Note**: When the input width is a non-multiple of 16, the encoder expects the application to pad the input frame to the nearest multiple of 16 to the right of the frame. In this case, application should set inputWidth to actual width but should provide the padded input YUV data buffer to encoder. The encoder then sets the difference of the actual width and padded width as crop information in the bit-stream. Default value = 720
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. This parameter is not supported, should be set equal to targetFrameRate. Default value = 25000
targetFrameRate	XDAS_Int32	Input	Target frame rate in fps * 1000. For example, if the frame rate is 30, set this field to 30000. Default value = 25000. Frame rate should be in multiple of 0.5 fps. Default value = 25000
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. Default value = 10000000.
intraFrameInterval	XDAS_Int32	Input	Interval between two consecutive intra frames. ❑ 0: First frame will be intra coded ❑ 1: No inter frames, all intra frames ❑ 2: Consecutive IPIPIP ❑ 3: 1PPIPPIPP or IPBIPBIPB, and so on Default value = 30
generateHeader	XDAS_Int32	Input	Encode entire access unit or only header. See XDM_EncMode enumeration for details. Default value = XDM_ENCODE_AU.

Field	Data type	Input/ Output	Description
captureWidth	XDAS_Int32	Input	Capture width parameter enables the application to provide input buffers with different line width (pitch) alignment than input width. For progressive content, if the parameter is set to: ❑ 0 - Encoded input width is used as pitch. ❑ >= encoded input width - capture width is used as pitch. For interlaced content, captureWidth should be equal to the pitch/stride value needed to move to the next row of pixel in the same field. Default value = 0
forceFrame	XDAS_Int32	Input	Force the current (immediate) frame to be encoded as a specific frame type. Only the following values are supported: ❑ IVIDEO_NA_FRAME - No forcing of any specific frame type for the frame. ❑ IVIDEO_I_FRAME - Force the frame to be encoded as I frame. ❑ IVIDEO_IDR_FRAME - Force the frame to be encoded as an IDR frame. Default value = IVIDEO_NA_FRAME.
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. This parameter is not supported. It should be set to 0.
mbDataFlag	XDAS_Int32	Input	Flag to indicate that the algorithm should use MB data supplied in additional buffer within inBufs. This parameter is not supported. It should be set to 0.

Note:

The following are the limitations on the parameters of IVIDENC1_DynamicParams data structure:

- inputHeight <= maxHeight
- inputWidth <= maxWidth
- refFrameRate <= maxFrameRate
- targetFrameRate <= maxFrameRate
- targetFrameRate should be multiple of 500
- The value of the refFrameRate and targetFrameRate should be the same

- APIs `refFrameRate` and `targetFrameRate` were initially maintained (XDM API perspective) to enable frame rate conversion by codec. For example, you could set `refFrameRate` = 30000 and `targetFrameRate` = 24000. This implies that the encoder will get input @ 30frames per sec and will convert frame rate from 30 to 24 while encoding. Hence, the encoded bit-stream will have only 24 frames of data per sec.
- DM365/DM368 implementation of `refFrameRate` and `targetFrameRate`: This feature is not supported in DM365/DM368. Hence, we make `refFrameRate = targetFrameRate`. For example:
  Capturing at 15 fps and required bitrate is 768kbps, set `refFrameRate = targetFrameRate` = 15000 and `targetBitrate` = 768000
  Capturing at 30fps and required bitrate is 1mbps, set `refFrameRate = targetFrameRate` = 30000 and `targetBitrate` = 1000000
  Capturing at 30fps to encode at 15fps with bitrate of 768kbps, Convert frame rate from 30 to 15 in application and then set `refFrameRate = targetFrameRate` = 15000 and `targetBitrate` = 768000
- `targetBitRate <= maxBitRate`
- The `inputHeight` and `inputWidth` must be multiples of two.
- The `inputHeight`, `inputWidth`, and `targetFrameRate` should adhere to the standard defined level limits. For an incorrect level, the encoder tries to match the best level for the parameters provided. However, if it exceeds level 5.0, an error is reported. As per the requirement, level limit can be violated for `targetBitRate`.
- When `inputHeight`/`inputWidth` are non-multiples of 16, encoder expects the application to pad the input frame to the nearest multiple of 16 at the bottom/right of the frame. In this case, application sets the `inputHeight`/`inputWidth` to the `actual height/actual width`; however, it should provide the padded input YUV data buffer to the encoder.
- When `inputWidth` is non-multiple of 16, the encoder expects capture width as padded width(nearest multiple of 16). If the capture width is 0, then the capture width is assumed to be the padded width. In all other cases, the capture width provided through input parameter is used for input frame processing.
- For out of bound and invalid parameters, encoder returns with fatal error.
- `intraFrameInterval` is used to signal the I frame interval in H.264. There is one more field in extended dynamic params called `idrFrameInterval`, which specifies the IDR frame interval for H.264. With each IDR frame, SPS and PPS is sent. The first frame of the sequence is always an IDR frame

4.2.1.11 IVIDENC1_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 encoded bit stream frames. This is useful when frames require buffering (for example, B frames), and to support buffer management. When there is no re-ordering, IVIDENC1_OutArgs::outputID will be the same as this inputID field. Zero (0) is not a supported inputID. This value is reserved for cases when there is no output buffer provided.
topFieldFirstFlag	XDAS_Int32	Input	Flag to indicate the field order in interlaced content. Valid values are XDAS_TRUE and XDAS_FALSE. This field is only applicable to the input image buffer. This field is only applicable for interlaced content and not progressive. Currently, supported value is XDAS_TRUE.

4.2.1.12 IVIDENC1_Status

Description

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

Fields

Field	Data type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
extendedError	XDAS_Int32	Output	Extended error code. See XDM_ErrorBit enumeration for details.
data	XDM1_SingleBufDesc	Input/Output	Buffer descriptor for data passing
bufInfo	XDM_AlgBufInfo	Output	Input and output buffer information. See XDM_AlgBufInfo data structure for details.

4.2.1.13 IVIDENC1_OutArgs

Description

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

Fields

Field	Data type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
extendedError	XDAS_Int32	Output	Extended error code. See XDM_ErrorBit enumeration for details.
bytesGenerated	XDAS_Int32	Output	The number of bytes generated.
encodedFrameType	XDAS_Int32	Output	Frame types for video. See IVIDEO_FrameType enumeration for details. Following values are only supported ❑ IVIDEO_I_FRAME ❑ IVIDEO_IDR_FRAME ❑ IVIDEO_P_FRAME ❑ IVIDEO_II_FRAME ❑ IVIDEO_PP_FRAME
inputFrameSkip	XDAS_Int32	Output	Frame skipping modes for video. See IVIDEO_SkipMode enumeration for details.

Field	Data type	Input/ Output	Description
outputID	XDAS_Int32	Output	Output ID corresponding to the encoder buffer. This can also be used to free the corresponding image buffer for further use by the client application code. In this encoder, outputID is set to IVIDENC1_InArgs::inputID.
encodedBuf	XDM1_SingleBufDesc	Output	The encoder fills the buffer with the encoded bit-stream. In case of sequences with only I and P frames, these values are identical to outBufs passed in IVIDENC1_Fxns::process() The encodedBuf.bufSize field returned corresponds to the actual valid bytes available in the buffer. The bit-stream is in encoded order. The outputId and encodedBuf together provide information related to the corresponding encoded image buffer.
reconBufs	IVIDEO1_BufDesc	Output	Pointer to reconstruction buffer descriptor.

4.2.2 H.264 Encoder Data Structures

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

- IH264VENC_Params
- IH264VENC_DynamicParams
- IH264VENC_InArgs
- IH264VENC_Status
- IH264VENC_OutArgs
- IH264VENC_Fxns

4.2.2.1 IH264VENC_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, IVIDENC1_Params.

Fields

Field	Data type	Input/ Output	Description
videncParams	IVIDENC1_Params	Input	See IVIDENC1_Params data structure for details. The size parameter in videncParams is set to size of IH264VENC_Params structure by default while using extended parameters.
profileIdc	XDAS_Int32	Input	Profile identification for the encoder. The current version supports High Profile. The value must be set to 66(Base line profile), 77(main profile), 100(high profile). Default value = 100.
levelIdc	XDAS_Int32	Input	Level identification for the encoder. See IH264VENC_Level enumeration for details. Default value = IH264VENC_LEVEL_40.
aspectRatioX	XDAS_Int32	Input	X scale for Aspect Ratio. The value should be greater than 0 and co-prime with AspectRatioY. Default value = 1
aspectRatioY	XDAS_Int32	Input	Y scale for Aspect Ratio The value should be greater than 0 and co-prime with AspectRatioX. Default value = 1.

Field	Data type	Input/ Output	Description
pixelRange	XDAS_Int32	Input	Range for the luma and chroma pixel values ❑ 0 – Restricted Range ❑ 1 – Full Range (0-255) Default value = 1
meAlgo	XDAS_Int32	Input	This field is reserved
timeScale	XDAS_Int32	Input	Time resolution value for Picture Timing Information This should be greater than or equal to frame rate in fps. See Appendix A for more details. Default value = 150.
numUnitsInTicks	XDAS_Int32	Input	Units of Time Resolution constituting the single Tick See Appendix A for more details. Default value = 1.
enableVUIparams	XDAS_Int32	Input	Flag for Enable VUI Parameters ❑ Bit 0: Controls VUI params insertion in SPS. If 0 -> VUI is not inserted in SPS, 1-> VUI is inserted in SPS. The VUI message is generated internally by the codec based on RC and some other API parameters ❑ Bit 1: Controls IDR frame insertion in case of RC parameter change, If 0 -> IDR is inserted with change in RC parameters, 1-> IDR is not inserted with change in RC parameters Note: If enableBufSEI = 1, VUI param insertion condition is enabled i.e. Bit 0 is assumed to be 1. This enables insertion of VUI param as per above condition set
entropyMode	XDAS_Int32	Input	Flag for Entropy Coding Mode ❑ 0 – CAVLC ❑ 1 – CABAC Default value = 1. This tool is supported only in Main Profile and High Profile (profileIdc = 77 and 100)
transform8x8FlagIntraFrame	XDAS_Int32	Input	Flag for 8x8 Transform for I frame ❑ 0 – Disable ❑ 1 – Enable Default value = 1. This tool is supported only in High Profile (profileIdc = 100)

Field	Data type	Input/ Output	Description
transform8x8FlagInterFrame	XDAS_Int32	Input	Flag for 8x8 Transform for P frame ❑ 0 – Disable ❑ 1 – Enable Default value = 0. This tool is supported only in High Profile (profileIdc = 100)
seqScalingFlag	XDAS_Int32	Input	Flag for use of Sequence Scaling Matrix ❑ 0 – Disable ❑ 1 – Auto ❑ 2 – Low ❑ 3 – Moderate ❑ 4 – Reserved Default value = 1. This tool is supported only in High Profile (profileIdc = 100) Currently the behavior for input value of 4 will be same as 3 i.e. Moderate SM.
disableHDVICPeveryFrame	XDAS_Int32	Input	Reserved
encQuality	XDAS_Int32	Input	Flag for Encoder setting ❑ 0 – version 1.1, backward compatible mode, ❑ 2 – High speed mode(This is same as encodingPreset = XDM_HIFG_SPEED). Default value = 2 .
unrestrictedMV	XDAS_Int32	Input	This field is reserved. UMV is always ON in the encoder.
enableARM926Tcm	XDAS_Int32	Input	Flag for enabling/disabling usage of ARM926 TCM: ❑ 1 – Uses ARM926 TCM ❑ 0 – Does not use ARM926 TCM Default value = 0 This control is only active for encodingPreset = XDM_USER_DEFINED and encQuality = 0, For other encoder preset and mode, there is no user control over it. It is internally set to 1 and ARM926 TCM is always used.
enableDDRbuff	XDAS_Int32	Input	Flag for enabling/disabling usage of DDR instead of IMCOP buffers. ❑ 1 – Uses DDR instead of VICP buffers. ❑ 0 – Use VICP buffers. Default value = 0

Field	Data type	Input/ Output	Description
sliceMode	XDAS_Int32	Input	Mode for specifying slice size ❑ 0 – No multi-slice ❑ 1 – Reserved. ❑ 2 – number of MBs per slice ❑ 3 – number of Mb rows per slice Default value = 0
outputDataMode	XDAS_Int32	Input	Mode for specifying low latency interface ❑ 0 – Low latency enabled. Codec interface at NAL encoding granularity ❑ 1 – Low latency disabled. Codec interface at frame encoding level
sliceFormat	XDAS_Int32	Input	Output Nal unit encoding format ❑ 0 – Output data in NAL stream format ❑ 1 – Output data in Byte stream format

Note:

- Default values of extended parameters are used when size fields are set to the size of base structure `IVIDENC1_Params`.
- `aspectRatio` and `pixelRange` information is included in the bit-stream only when `enableVUIparams` is set to 1.
- When `enableVUIparams` is set to 2, IDR frame is not inserted when any of the following parameters are changed dynamically.
  i. `Framerate`
  ii. `Bitrate`
  iii. `MaxDelay`
  iv. `RC Algorithm.`

  When `enableVUIparams` is set to 0 or 1, an IDR frame containing SPS and PPS parameter is inserted in the stream.
- The behavior of `aspectRatioX` and `aspectRatioY` is similar to what is defined in the section E.1.3 of H.264 standard. You need to specify X and Y values. If it matches with the value as provided in table E-1, aspect_ratio_idc is sent in the streams. If it does not match, sar_width and sar_height is sent explicitly with aspect_ratio_idc set to 255(extended SAR)
- If the level is not set appropriately, the encoder tries to fit a correct level. However, if it exceeds level 5.0, an error is reported.
- If interlace encoding is enabled for levels less than 2,1 or level more than level 4.1 encoder will return fatal error during instance creation.
- When `encodingPreset = XDM_HIGH_SPEED/ XDM_HIGH_QULAITY` or `encQuality` = 2, Perceptual rate control feature is disabled in the current encoder version:
- Types of Multiple Slices supported in different modes:
  - Version 1.1, Backward comptible mode(`encQuality` = 0):

Multiple slices based on number of MBs per slice and number of rows per slice.

- Platinum mode Mode(`encQuality` = 2): Multiple slices based on number of rows per slice.

4.2.2.2 IH264VENC_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, `IVIDENC1_DynamicParams`.

Fields

Field	Data type	Input/ Output	Description
`videncDynamicParams`	`IVIDENC1_DynamicParams`	Input	See `IVIDENC1_DynamicParams` data structure for details. The size parameter of `DynamicParams` is set to size of `IVIDENC1_DynamicParams` structure by default while using extended parameters.
`intraFrameQP`	`XDAS_Int32`	Input	Quantization Parameter (QP) of I-frames in fixed QP mode. Valid value is 0 to 51. It is useful only when: ❑ `rateControlPreset` of `IVIDENC1_Params` is equal to `IVIDEO_NONE`. ❑ `RcAlgo` = 2 (Fixed QP) ❑ `targetBitRate` = 0 Default value = 28
`interPFrameQP`	`XDAS_Int32`	Input	Quantization Parameter (QP) of P-frames in fixed QP mode. Valid value is 0 to 51. It is useful only when: ❑ `rateControlPreset` of `IVIDENC1_Params` is equal to `IVIDEO_NONE`. ❑ `RcAlgo` = 2 (Fixed QP) ❑ `targetBitRate` = 0 Default value = 28
`initQ`	`XDAS_Int32`	Input	Initial Quantization (QP) for the first frame. Valid values include -1 and any value between 0 to 51. The parameter is applicable only when rate-control is enabled. Should be set based on the target bit-rate. Default value = 28 Recommended value = -1. When -1 is used, encoder calculates initial Qp based on bit rate, frame rate and input resolution. This calculated Qp value is used for first frame.

Field	Data type	Input/ Output	Description
rcQMax	XDAS_Int32	Input	Maximum value of Quantization Parameter (QP) to be used while encoding. Valid value is 0 to 51. The value for rcQMax should not be less than rcQMin. The parameter is applicable only when rate-control is enabled. Default value = 45
rcQMin	XDAS_Int32	Input	Minimum value of Quantization Parameter (QP) to be used while encoding. Valid value is 0 to 51. The value for rcQMin should not be greater than rcQMax. The parameter is applicable only when rate-control is enabled. Default value = 0.
rcQMaxI	XDAS_Int32	Input	Maximum value of Quantization Parameter (QP) to be used while encoding Intra Frame. Valid value is 0 to 51. The value for rcQMaxI should not be less than rcQMinI. The parameter is applicable only when rate-control is enabled. Default value = 42
rcQMinI	XDAS_Int32	Input	Minimum value of Quantization Parameter (QP) to be used while encoding Intra Frame. Valid value is 0 to 51. The value for rcQMinI should not be greater than rcQMaxI. The parameter is applicable only when rate-control is enabled. Default value = 0.
airRate	XDAS_Int32	Input	Parameter for forced Intra MB insertion in P-frames. ❑ 0 – No forced Intra MBs ❑ n > 0 – number of forced Intra MB in each frame. Default value = 0. This feature is not supported for interlaced content.
sliceSize	XDAS_Int32	Input	The interpretation of sliceSize depends on sliceMode value. See the note at end of 4.2.2.2 for details on sliceSize range and interpretation.
lfDisableIdc	XDAS_Int32	Input	Option to enable or disable loop filter ❑ 0 – Loop Filter Enable ❑ 1 – Loop Filter Disable ❑ 2 – Disable Filter across slice boundaries Default value = 0
rcAlgo	XDAS_Int32	Input	Option to specify the type of Rate Control Algorithm ❑ 0 – CBR ❑ 1 – VBR ❑ 2 – Fixed QP CBR Rate Control algorithm is not supported for interlaced encoding and will be automatically disabled by encoder. Default value = 1

Field	Data type	Input/ Output	Description
maxDelay	XDAS_Int32	Input	Maximum acceptable delay in milliseconds for rate control. ❑ Min Limit: No minimum value check ❑ Max Limit : 10000 ms It is recommended to use value greater than 100 ms. Typical value is 1000 ms. By default, this is set to 2000 ms at the time of encoder object creation.
intraSliceNum	XDAS_Int32	Input	This field is reserved
meMultiPart	XDAS_Int32	Input	This field is reserved
enableBufSEI	XDAS_Int32	Input	Flag for enabling buffering period SEI message ❑ 0 – Disable ❑ 1 – Enable Default value = 0 Buffering period SEI insertion is not supported for interlaced content
enablePicTimSEI	XDAS_Int32	Input	Flag for enabling picture timing SEI message ❑ 0 – Disable ❑ 1 – Enable This parameter is disabled if EnableBufSEI is disabled. Default value = 0 Picture Timing SEI insertion is not supported for interlaced content
intraThrQF	XDAS_Int32	Input	This field is reserved.
perceptualRC	XDAS_Int32	Input	Flag for enabling perceptual QP modulation of MBs ❑ 0 – Disable ❑ 1 – Enable Default value = 1 This feature is only present if encQuality = 0 under encodingPreset = XDM_USER_DEFINED. For XDM_HIGH_SPEED and XDM_HIGH_QUALITY, this feature is disabled. PRC is disable automatically for maxDelay<1000 and rcAlgo = CBR
idrFrameInterval	XDAS_Int32	Input	Interval between two consecutive IDR frames ❑ 0: first frame will be IDR coded ❑ 1: No inter frames, all IDR frames ❑ 2: Consecutive IDR P IDR P ❑ 3: IDR P P IDR P P IDR .. or IDR P B IDR P B IDR P B ….and so on Default value = 0.

Field	Data type	Input/ Output	Description
mvSADoutFlag	XDAS_Int32	Input	This flag enables dumping of MV and SAD value of the encoded stream. If the flag is enabled, XDM_GETBUFINFO call will request for one extra buffer to dump the MV and SAD. See note for details. Default value = 0.
resetHDVICPeveryFrame	XDAS_Int32	Input	Flag to reset HDVICP at the start of every frame that is encoded. This is useful for multi-channel and multi-format encoding. ❑ 1 – ON ❑ 0 – OFF Default value = 1. If this flag is set, H.264 encoder assumes that the memories of HDVICP was overwritten by some other codec or by other instance of same codec with different quality settings between process call and hence reloads the code and data. For example : Application will set this flag to 1 if running another instance of different codec like H264 decoder or if running another H264 encoder instance with different quality setting in encQuality or encodingPreset. However, application can set this flag to 0 for better performance if it runs multiple instances of H264 encoder with same quality settings in encQuality and encodingPreset.
enableROI	XDAS_Int32	Input	Flag to enable/disable ROI coding. ❑ 1 – enable ROI coding. ❑ 0 – disable ROI coding. Default value = 0. This flag will be automatically disabled when rcAlgo = Fixed QP.

Field	Data type	Input/ Output	Description
metaDataGenerateConsume	XDAS_Int32	Input	Flag to enable/disable metaData Consume and generate. ❑ 0: Not used. ❑ 1: Generate metaData in the current instance. ❑ 2: Consume metaData in the current instance. ❑ 3: metaData genenrated but not yet consumed. Default value = 0. When this flag value is set to 1, the encoder will generate the metadata and store the frame related information in the FrameInfo_Interface structure. This structure is then passed to the application. If the flag value is 2 then the current encoder instance will use the metadata generated by other encoder to improve/customise the encoding operation. If the flag value is 3 the metaData is generated by the low resolution encoder but not yet consumed by high resolution encoder. (See Appendix D for detailed usage).
disableMVDCostFactor	XDAS_Int32	Input	Reserved
putDataGetSpaceFxn	IH264VENC_TI_DataSyncPutGetFxn	Input	Pointer to callback module required to enable low latency feature
dataSyncHandle	IH264VENC_TI_DataSyncHandle	Input	Handle to DataSync descriptor

Note:

- enablePicTimSEI values are used only when enableBufSEI is set to 1.
- rcAlgo values are used only when IVIDENC1_Params ->RateControlPreset = IVIDEO_USER_DEFINED.
- rcQMax, rcQMin, initQ, and maxDelay values are used only when the encoder does not run in fixed QP mode.
- Generally idrFrameInterval will be larger than intraFrameInterval. For example, idrFrameInterval = 300

and `intraFrameInterval` = 30. This means that at every 30th frame, there will be an I frame. But at every 300th frame, an IDR frame will be placed instead of I frame. IDR frame is used for synchronization.

- The MV and SAD is dumped in the `outBuf`. The extra buffer is requested during `XDM_GETBUFINFO` call. If multiple slice is on, then MV-SAD information is in the index 2 of the buffers pointed by `XDM_BufDesc *outBufs` and index 1 is for packet size information. If multiple slice is off, the MV-SAD is dumped in index 1 of buffer pointers. Index 0 is always used for bit-stream data. MV SAD information is in the following format:
  - Word0: MVy[bit 31-16]:MVx[bit 15-0]
  - Word1: SAD [bit 31-0]

  For motion vector and SAD, the top left partition is used in case multiple MV is enabled.
- Regions where the viewer pays more attention to are called regions of interest (ROI). In such scenarios it is important that the ROI areas are reproduced as reliable as possible since they contribute significantly towards the overall quality and perception of the video. This is achieved by assigning higher number of bits to the ROI areas when compared to non-ROI areas.
- If the current frame at low resolution encoder is encoded as IDR/I frame then no scene change information is passed to high resolution encoder.
- Forcing intra MBs when `airRate`>0 is done as explained below.

  Randomized AIR is used as intra refresh startegy. In this case atlease `airRate` number of MBs in a frame will be set as intra, except for the last module. There could be more than `airRate` MBs as intra because there could be macroblocks coded as intra due to intra/inter mode decision.

  Consider that there are 396 MBs in a frame and airRate = 10. So after 39 frames 390 MBs will be refreshed. So for 40th frame only 6 MBs get refreshed ti intra. So for all frames atlease airRate number of MBs in a frame will not be Intra.

  For `encQuality` = 0, when AIR is ON, contrained intra prediction gets used. In other modes of operation, contrained intra prediction is not used when AIR is ON.
- If `sliceMode` = 0 then `sliceSize` value is ignored. Entire frame will be encoded as a single slice.
- `SliceMode` = 1 is Reserved
- If `SliceMode` = 2 then sliceSize indicates:
  Size of each slice in number of MBs.
  - 0 – Single Slice per Frame
  - >0 – Multiple Slices with each slice having MBs <= sliceSize.

  Default value = 0

  This feature is only present when `encQuality` = 0 .
  Slicesize value should be multiple of 2 always. Value of slice size is limited by total number of MBs in frame.

  In case of inputs having odd multiple of MBs in a row, an virtual MB

is considered, For example, for an input with 11MBs/row, if user wants 1row/slice;then sliceSize should be 12(11+1virtualMB=12). User should take care of accounting this virtual MB while setting sliceSize.

- If `SliceMode` = 3 then `sliceSize` indicates:
  Size of each slice in number rows per slice.
  - 0 – Single Slice per Frame
  - >0 – Multiple Slices with each slice having rows = sliceSize.

  Default value = 0

  This feature is supported in all modes.
  Value of slice size is limited by total number of rowss in frame.

4.2.2.3 IH264VENC_InArgs

Description

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

Fields

Field	Data type	Input/ Output	Description
`videncInArgs`	`IVIDENC1_InArgs`	Input	See `IVIDENC1_InArgs` data structure for details.
`timeStamp`	`XDAS_Int32`	Input	Time stamp value of the frame to be placed in bit stream. This should be integral multiple of `TimerResolution`/ (frame rate in fps). Initial time stamp value (for first frame) should be 0. Default is calculated as Frame number * `TimerResolution`/ (Frame rate in fps). See Appendix A for more details.
`insertUserData`	`XDAS_Int32`	Input	Flag to enable insertion of user data as part of SEI unregistered user data
`lengthUserData`	`XDAS_Int32`	Input	Length of user data to be inserted in the bit-stream. The codec will create space in bit-stream of the given length for user data insertion.
`roiParameters`	`ROI_Interface`	Input	This is to pass the ROI related data to the algorithm. See ROI_Interface data structure under section 4.3 for details.
`numOutputDataUnits`	`XDAS_Int32`	Input	This specifies number of NAL units which encoder will encode before triggering call back API . For details, See section 4.5

Note:

TimeStamp is included only when IH264VENC_DynamicParams->EnablePicTimSEI is set to 1.

4.2.2.4 IH264VENC_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, IVIDENC1_Status.

Fields

Field	Data type	Input/ Output	Description
videncStatus	IVIDENC1_Status	Input/Output	See IVIDENC1_Status data structure for details.

4.2.2.5 IH264VENC_OutArgs

Description

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

Fields

Field	Data type	Input/ Output	Description
videncOutArgs	IVIDENC1_OutArgs	Output	See IVIDENC1_OutArgs data structure for details.
numPackets	XDAS_Int32	Output	Total number of packets/slices in the encoded frame. The size of the packet is part of outBufs memory of the process call.
offsetUserData	XDAS_Int32	Output	This is the offset in the bit-stream for user data insertion. The offset (bytes) is with respect to the output buffer where the encoded frame is dumped after the process() call. Application should move to this offset and place the user data of lengthUserData. Codec only adds placeholder in bit-stream for user data insertion. Actual user data insertion has to be done by the application.

4.2.2.6 IH264VENC_Fxns

Description

This structure defines all of the operations for the H.264 Encoder instance object.

Fields

Field	Data type	Input/ Output	Description
ividenc	IVIDENC1_Fxns	Output	See IVIDENC1_Fxns data structure for details.

4.3 H.264 Encoder ROI specific Data Structures and Enumerations

This section includes the following H.264 Encoder ROI specific structures and enumerations:

- `XDM_Point structure.`
- `XDM_Rect structure.`
- `ROI_type enumeration.`
- `ROI_Interface structure.`

4.3.1.1 XDM_Point

‖ Description

This structure defines all the fields required to specify location of point. This will be used to specify X and Y co-ordinates of given point.

‖ Fields

Field	Data type	Input/ Output	Description
x	`XDAS_Int32`	Input	This will specify the X co-ordinate of a given point.
y	`XDAS_Int32`	Input	This will specify the Y co-ordinate of a given point.

4.3.1.2 XDM_Rect

‖ Description

This structure defines all the fields required to specify a rectangle. This will be used to specify top left and bottom right co-ordinates of a given ROI.

‖ Fields

Field	Data type	Input/ Output	Description
`topLeft`	`XDM_Point`	Input	This will specify the X and Y co-ordinate of top left point of given ROI. See `XDM_Point` data structure for details.

Field	Data type	Input/ Output	Description
bottomRight	XDM_Point	Input	This will specify the X and Y co-ordinate of bottom right point of given ROI. See XDM_Point data structure for details.

4.3.1.3 ROI_type

Description

This enumeration defines all the different types of ROI.

Fields

Enumeration Class	Symbolic Constant Name	Description
ROI_type	FACE_OBJECT	Type of ROI is FACE OBJECT
	BACKGROUND_OBJECT	Type of ROI is BACKGROUND OBJECT
	FOREGROUND_OBJECT	Type of ROI is FOREGROUND OBJECT
	DEFAULT_OBJECT	Type of ROI is DEFAULT OBJECT
	PRIVACY_MASK	Type of ROI is PRIVACY MASK

4.3.1.4 ROI_Interface

Description

This structure defines all the fields required to send ROI data to the algorithm.

Field	Data type	Input/ Output	Description
listROI [MAX_ROI]	XDM_Rect	Input	For a given ROI, this gives the X and Y co-ordinates of the top left and bottom right points. See XDM_Rect data structure for details.
roiType [MAX_ROI]	ROI_type	Input	This field specifies the type of ROI. Codec may take some special action depending on type of ROI. See ROI_type enumeration for details.

Field	Data type	Input/ Output	Description
numOfROI	XDAS_Int32	Input	Number of ROI limited by MAX_ROI.
roiPriority [MAX_ROI]	XDAS_Int32	Input	Priority of the given ROI. Valid values include all integers between -4 and 4. . 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.

Note:

- In current implementation, MAX_ROI supported is 5.
- There is support for different priorities for different ROIs in this version of H264 Encoder. But ROIs of same ROI_type should have same priority.
- Overlapping of ROIs of same ROI_type is allowed in this release.
- ROI of type PRIVACY_MASK is not supported in this version of H264 Encoder.
- ROI can be of any type as mentioned in ROI_type. 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.

4.4 H264 Encoder Two Pass Encoder data structure

In simple two pass encoding following data structures have been used

- `MBinfo Structure`
- `MBRowInfo Structure`
- `FrameInfo_Interface Structure`

4.4.1 MBinfo

‖ Description

This structure is used to store MB information. It contains following elements.

‖ Fields

Field	Data type	Input/ Output	Description
`numBitsMB`	`XDAS_UInt16`	Output	Number of bits to encode MB
`mbCodingMode`	`XDAS_UInt8`	Output	MB coding mode Inter or Intra
`mbQP`	`XDAS_UInt8`	Output	QP of MB

4.4.2 MBRowinfo

‖ Description

This structure contains buffer description of MB row related parameters.

‖ Fields

Field	Data type	Input/ Output	Description
`gmvVert`	`XDAS_UInt32`	Output	GMV information per row.

4.4.3 Frameinfo_Interface

Description

This Structure contains buffer description of frame related Parameters which are pass from low resolution encoder to high resolution encoder.

Fields

Field	Data type	Input/ Output	Description
`Width`	`XDAS_UInt16`	Output	Width of the frame in pixels.
`Height`	`XDAS_UInt16`	Output	Height of the frame in pixels.
`sceneChangeFlag`	`XDAS_UInt32`	Output	Flag to indicate scene change observed at low-resolution encoder level.
`bitsPerFrame`	`XDAS_UInt32`	Output	Number of bits used to encode frame by low-resolution encoder.
`frameRate`	`XDAS_UInt32`	Output	Frame rate per second.
`Bitrate`	`XDAS_UInt32`	Output	Target bit rate in bps.
`mvSADpointer`	`XDAS_UInt32 *`	Output	Pointer to MVSAD of all the MBs in a frame.
`mbComplexity`	`MBinfo *`	Output	Pointer to MB information of all the MBs in a frame.
`gmvPointerVert`	`MBRowinfo *`	Output	Pointer to vertical GMV values per row.

Note:

- When `mvSADflag` is disabled the `mvSADpointer` points to NULL.
- In current implementation we are not populating MBinfo and MBRowinfo structures. Currrently, `mbComplexity and gmvPointerVert` pointers points to NULL.
- When the scenechange is detected at low resolution encoder, IDR frame is forced at high resolution encoder at the corresponding frame number.

4.5 H.264 Encoder Low latency specific Data Structures and Enumerations

This section includes the following H.264 Encoder Low Latency specific structures, constant, typedefs and enumerations:

- IH264VENC_TI_DataSyncDesc.
- IH264VENC_TI_MAXNUMBLOCKS
- IH264VENC_TI_DataSyncHandle
- IH264VENC_TI_DataSyncPutGetFxn
- IH264VENC_TI_DataMode
- IH264VENC_TI_SliceFormat enumeration.

4.5.1 Structures

4.5.1.1 IH264VENC_TI_DataSyncDesc

Description

This structure is a descriptor for the chunk of data being transferred via callback for producing the encoded data at NAL level

Fields

Field	Data type	Input/ Output	Description
Size	XDAS_Int32	Input	Size of this structure.
numBlocks	XDAS_Int32	Input	Number of blocks provided for writing the encoded NAL. Valid values are between 1 to IH264VENC_TI_MAX_NUMBLOCKS
varBlockSizesFlag	XDAS_Int32	Input	Flag indicating whether any of the data blocks vary in size. Valid values are XDAS_TRUE and XDAS_FALSE. **Current supported value is XDAS_FALSE.**
baseAddr	XDAS_Int32 *	Input	Array of pointers to the first byte of all (numBlocks) blocks provided for writing the encoded slice.
blockSizes	XDAS_Int32*	Inout	This array contains the sizes of each valid blocks

4.5.2 Constant

4.5.2.1 IH264VENC_TI_MAXNUMBLOCKS

Description

This MACRO defines max value of numBlocks accepted by encoder when operated in IH264VNC_TI_SLICEMODE outputData mode

4.5.3 Typdef

4.5.3.1 IH264VENC_TI_DataSyncHandle

Description

This typedefs is handle that identifies DataSync FIFO. Fields

Field	Data type	Input/ Output	Description
IH264VENC_TI_DataSyncHandle	Void *	Input	This handle is provided by the application to handle DataSync Fifo. Encoder passes this handle back to application when providing output data via callback

4.5.3.2 IH264VENC_TI_DataSyncPutGetFxn

Typedef to pointer to callback module used by encoder to signal "data ready" to consumer and to get the space for next set of data. Consumer need to define this API. Returns the successor failure status. Valid return value is XDM_EOK or XDM_EFAIL.

Name

IH264VENC_TI_DataSyncPutGetFxn.

Synopsis

```
typedef XDAS_Int32 (*
IH264VENC_TI_DataSyncPutGetFxn)(IH264VENC_TI_DataSyncHandl
e dataSyncHandle, IH264VENC_TI_DataSyncDesc dataSyncDesc);
```

Arguments

```
IH264VENC_TI_DataSyncHandle dataSyncHandle /* Handle of
dataSync provided by application */
```

Arguments

```
IH264VENC_TI_DataSyncDesc *dataSyncDesc /*
dataSyncDescriptor containing encoded slice to be provided
to application */
```

Return Value

```
XDAS_Int32 /* Return Status – XDM_EOK/XDM_EFAIL */
```

4.5.4 Enum

4.5.4.1 IH264VENC_TI_DataMode

Description

This enumeration is used to specify codec when to provide encoded data – after entire frame encoding or after slice encoding.

Fields

Enumeration Class	Symbolic Constant Name	Description
IH264VENC_TI_DataMode	IH264VENC_TI_SLICEMODE	provide encoded data after one slice is encoded
	IH264VENC_TI_ENTIREFRAME	provide encoded data after entire frame is encoded

4.5.4.2 H264VENC_TI_SliceFormat

Description

Describes the output slice format of encoder. This enumeration type is used to specify codec encode stream format type .

Fields

Enumeration Class	Symbolic Constant Name	Description
IH264VENC_TI_DataMode	IH264VENC_TI_NALSTREAM	Output data in NAL stream format
	IH264VENC_TI_BYTESTREAM	Output data in BYTE stream format

Notes:

- If the `outBuf` is cacahed, then the application needs to take care of cacahe invalidating the data before doing any read/write operation. This is because the input/output data is always read through DMA and not CPU.

Example Usage:

- Configuring encoder

  Assume slice size as 2MB row. Set encoder with below parameters:

```
IH264VENC_Params->sliceMode          = 3
IH264VENC_Params->outputDataMode     = 0
IH264VENC_Params->sliceFormat        = 1
                    (assuming byte stream encoding)
IH264VENC_DynamicParams->sliceSize   = 2
IH264VENC_InArgs->numOutputDataUnits = 1
```

  This will enable encoder to produce slice of 2MB row and the Low latency call back API will get called after 1 slice encode for data exchange.

- Syncronization and Data Exchange

  If the encoder is run in the above mode, the application will see the call back function getting invoked after 1 slice encode. Application can use this call back API for synchronization as well as data exchange. If the next 2MB row of data is put into DDR by capture driver, application can give the next output slice pointer to the codec and release the call back. This will make encoder proceed with further encoding. Please note that we use output slice buffer pointer of encoded bitstream rather than input YUV to control the encoder. The input YUV pointer is give at the start of process call only as in the normal encoding.

4.6 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()`, `algDeactivate()`
- **Termination** – `algFree()`

You must call these APIs in the following sequence:

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

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

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

4.6.1 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 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()`

4.6.2 Initialization API

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

‖ Name

`algInit()` – initialize an algorithm instance

‖ Synopsis

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

‖ Arguments

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

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

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

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

‖ Return Value

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

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

‖ Description

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

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

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

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

The last argument is a pointer to a structure that defines the algorithm initialization parameters.

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

‖ See Also

`algAlloc()`, `algMoved()`

4.6.3 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 `DynamicParams` data structure (see Data Structures section for details).

‖ Name

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

‖ Synopsis

```
XDAS_Int32 (*control) (IVIDENC1_Handle handle,
IVIDENC1_Cmd id, IVIDENC1_DynamicParams *params,
IVIDENC1_Status *status);
```

‖ Arguments

```
IVIDENC1_Handle handle; /* algorithm instance handle */

IVIDENC1_Cmd id; /* algorithm specific control commands*/

IVIDENC1_DynamicParams *params /* algorithm run-time
parameters */

IVIDENC1_Status *status /* algorithm instance status
parameters */
```

‖ Return Value

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

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

‖ Description

This function changes the run-time parameters of an algorithm instance and queries the algorithm's status. `control()` must only be called after a successful call to `algInit()` and must never be called after a call to `algFree()`.

The first argument to `control()` is a handle to an algorithm instance.

The second argument is an algorithm specific control command. See `XDM_CmdId` enumeration for details.

The third and fourth arguments are pointers to the `IVIDENC1_DynamicParams` and `IVIDENC1_Status` data structures respectively.

Note:

The control API can be called with base or extended `DynamicParams`, and `Status` data structure. 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()`.
- `handle` must be a valid handle for the algorithm's instance object.

‖ Post conditions

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

- If the control operation is successful, the return value from this operation is equal to `IALG_EOK`; otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.
- If the control command is not recognized, the return value from this operation is not equal to `IALG_EOK`.

‖ Example

See test application file, h264encoderapp.c available in the \client\test\src sub-directory.

‖ See Also

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

4.6.4 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 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 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)(IVIDENC1_Handle handle,
IVIDEO1_BufDescIn *inBufs, XDM_BufDesc *outBufs,
IVIDENC1_InArgs *inargs, IVIDENC1_OutArgs *outargs);
```

‖ Arguments

```
IVIDENC1_Handle handle; /* algorithm instance handle */

IVIDEO1_BufDescIn *inBufs; /* algorithm input buffer
descriptor */

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

IVIDENC1_InArgs *inargs /* algorithm runtime input arguments
*/

IVIDENC1_OutArgs *outargs /* algorithm runtime output
arguments */
```

‖ Return Value

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

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

‖ Description

A call to function initiates the encoding/decoding process for the current frame.

The first argument to `process()` is a handle to an algorithm instance.

The second and third arguments are pointers to the input and output buffer descriptor data structures respectively (see `XDM_BufDesc` data structure for details). Input/output buffers should be allocated in non-cacheable /non-bufferable region if low latency is enabled.

The fourth argument is a pointer to the `IVIDENC1_InArgs` data structure that defines the run-time input arguments for an algorithm instance object.

The last argument is a pointer to the `IVIDENC1_OutArgs` data structure that defines the run-time output arguments for an algorithm instance object.

In case of interlaced content, process call has to be invoked for each field.

> **Note:**
>
> The `process()` API can be called with base or extended `InArgs` and `OutArgs` data structures. 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()`.
- `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.

Post conditions

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.

Example

See test application file, h264encoderapp.c available in the \client\test\src sub-directory.

See Also

`algInit()`, `algDeactivate()`, `control()`

Note:

- A video encoder or decoder cannot be pre-empted 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 YUV 4:2:0 SP. The encoder output is H.264 encoded bit stream.

‖ 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()`

4.6.5 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()`

> **Note:**
>
> In the current implementation, `algFree()` API additionally resets HDVICP hardware co-processor and also releases DMA resources held by it. Thus, it is important that this function is used only to release the resource at the end and not in between `process()`/`control()` API functions.

This page is intentionally left blank

Appendix A

Time-Stamp Insertion

The DM365/DM368 H.264 Encoder supports insertion of frame time-stamp through the Supplemental Enhancement Information (SEI) Picture Timing message. The time-stamp is useful for audio-synchronization and determining the exact timing for display of frames. The parameters coded in the SEI Picture Timing Message are also useful for testing HRD compliance.

The application should take proper care while setting the parameters for time-stamp and the actual time-stamp for each frame. Ideally, the time-stamp can be set based on the frame-rate. This simplifies the process of generating time-stamps. However, the application is free to use any method of time-stamp generation.

Time-stamp based on frame-rate can be generated as follows.

Let f be the frame-rate of the sequence. Assuming a constant frame-rate sequence, set

```
TimeScale = k * f

NumUnitsinTicks = n

where k is an integer such that (k * f) and (k/n) are
integers

units_per_frame = k/n
```

For the first frame, set the `TimeStamp` parameter in `inArgs` structure to 0. For the subsequent frames, increment the `TimeStamp` by `units_per_frame`

Example 1.

```
f = 30.

Let k = 2

TimeScale = 2 * 30 = 60

NumUnitInTicks = 1

units_per_frame = 2

TimeStamp = 0, 2, 4, 6, 8…
```

Example 2.

```
f = 25

k = 2

TimeScale = 2 * 25 = 50

NumUnitsInTicks = 2

units_per_frame = 1

TimeStamp = 0, 1, 2, 3, 4…
```

Example 3.

```
f = 15

k = 1000

TimeScale = 1000 * 15 = 15000

NumUnitsInTicks = 1000

units_per_frame = 1

TimeStamp = 0, 1, 2, 3, 4…
```

Example 4.

```
f = 0.5

k = 200

TimeScale = 200 * 0.5 = 100

NumUnitsinTicks = 100

units_per_frame = 2

TimeStamp = 0, 2, 4, 6, 8
```

Appendix B

Error Description

`Encoder_Create()` returns FATAL_ERROR for out of range/invalid input parameter values. Also, the unsupported features usage in the profiles will also result in FATAL_ERROR. List of unsupported features with respect to the profile is listed in the following table.

ProfileIDC	Profile	Inputparam values that results in FATAL ERROR

66	Baseline	`inputContentType =1`
		`transform8x8FlagIntraFrame=1`
		`transform8x8FlagInterFrame=1`
		`seqScalingFlag=1`
		`entropyMode=1`

77	Main	`transform8x8FlagIntraFrame=1`
		`transform8x8FlagInterFrame=1`
		`seqScalingFlag=1`

100	High	-

In addition to this, if any other input parameter is beyond the range specified in the user guide, it will result in codec create or control API failure.

This page is intentionally left blank

Appendix C

VICP Buffer Usage By Codec

H.264 codec uses VICP buffers for its internal encode/decode operation. This buffer is accessed using EDMA. This section describes in brief how the buffers are used.

The Framework component (FC) manages the VCIP buffers using VCIP resource manager. In context of DM365/DM368, VICP buffers can be used by following algorithms:

- MPEG4 and JPEG running on MJCP
- H.264 codec running on HDVICP
- Preprocessing algorithms or noise filter running on IMX/NSF

Any of these algorithms can place its request to VICP buffers. FC services the VICP buffer request in a sequential manner.

	[Bytes]	Byte Address
iMX IMG Buf A	4096	0x00000 - 0x00FFF
iMX IMG Buf B	4096	0x01000 - 0x01FFF
iMX IMG Buf C	4096	0x02000 - 0x02FFF
iMX IMG Buf D	4096	0x03000 - 0x03FFF
iMX IMG Buf E	4096	0x04000 - 0x04FFF
iMX Coef 0	16384	0x05000 - 0x08FFF
iMX Coef 1	16384	0x09000 - 0x0CFFF
iMX Cmd 0	4096	0x0D000 - 0x0DFFF
iMX Cmd 1	4096	0x0E000 - 0x0EFFF
Sequencer PMEM	4096	0x0F000 - 0x0FFFF
		Register + Reserved
biMX cmd	1024	0x16000 - 0x163FF
biMX Org.	2048	0x16400 - 0x16BFF
biMX Ref.	8192	0x16C00 - 0x18BFF
QiQ Mem	1024	0x18C00 - 0x18FFF
DC/AC Pred Mem	256	0x19000 - 0x191FF
MV Mem.	512	0x91200 - 0x193FF
Huff Mem.	4096	0x19400 - 0x1A3FF
Seq Buf #1	4096	0x1A400 - 0x1B3FF
Seq Buf #2	4096	0x1B400 - 0x1C3FF
Seq Buf #3	2048	0x1C400 - 0x1CBFF
Seq Buf #4	2048	0x1CC00 - 0x1D3FF
Sequencer DMEM	4096	0x1F000 - 0x1F3FF

Figure C-1. VICP Buffers Managed By FC.

The above diagram shows the buffers of VICP managed by FC. The memories shaded in green are managed by FC. The memories in red are reserved for MJCP only.

FC gives the VICP memory to the algorithm from the start of the pool. Hence, it is the application's responsibility to instantiate the various algorithms in a way that an efficient usage of VICP buffers is achieved.

The amount of VICP buffer usage by the codec is part of datasheet provided in the release

Appendix D

ARM926 TCM Buffer Usage By Codec

H.264 encoder uses ARM926 TCM buffers for its internal encode operation. This buffer is accessed using EDMA. This section briefly describes the buffer usage.

The ARM926 processor provides a complete high performance sub-system, which includes separate instruction, data, tightly-coupled memories (TCMs) and internal RAM interfaces. Instruction and data access is differentiated by accessing different memory map regions, with the instruction region from 0x0000 through 0x7FFF and data from 0x10000 through 0x17FFF.

In context of DM365/DM368, ARM926 DTCM can be used for the following:

- ARM926 for system level usage
- H.264 codec running on HDVICP

The reason for H.264 codec running in HDVICP to use ARM926 TCM:

As seen in Appendix C, H.264 codec uses part of VICP buffers for its execution. However, when this codec is run along with an application that requires more VICP buffers (like MPEG4 and JPEG running on MJCP), then algorithm cannot use the VICP buffers originally used by it. Therefore, some of the buffers used in VICP will be transferred to ARM926 TCM. The ARM926 TCM buffers are managed by Framework component (FC) using ARM TCM resource manager.

The user can indicate his choice of using ARM926 TCM by suitably setting the create time parameter `useARM926Tcm` in `config` file.

Setting the `useARM926Tcm` to

- 0 - Do not use ARM926 TCM
- 1 - Use ARM926 TCM. This is supported for widths up to `maxWidth` of 1280.

The amount of ARM926 TCM buffer usage by the codec is a part of datasheet provided in the release.

The ARM926TCM memory in Linux is managed though CMEM pools, Hence, the below pool allocation needs to be appended when ARM926TCM is used.

- `allowOverlap`=1 phys_start_1=0x00001000 phys_end_1=0x00008000 pools_1=1x28672

This page is intentionally left blank

Appendix E

Simple Two-pass Encoding Sample Usage

Multi-pass encoding can be used to improve the quality of the H264 encoded video. This version of H264 encoder on DM365/DM368 supports simple two pass (STP) encoder. In STP encoder, two encoders run sequentially for every frame captured, first the low-resolution encoder and then the high-resolution encoder.

The low-resolution encoder accumulates the frame specific information (metadata) in the structure described in the Section 4.4. After completion of low pass encoding, metadata is passed to the high-resolution encoder. The high-resolution encoder uses metadata appropriately to improve the quality of the encoded video

Various example cases of simple two encoding are given below:

1) **Case-1:** If both encoder runs at same frame rate and no frame skip occurs; generation and consumption of the frame level information (metadata) happens for each frame.

2) **Case-2:** If there is a frame skip at low-resolution encoder; no information is passed to high-resolution encoder for the corresponding frame.

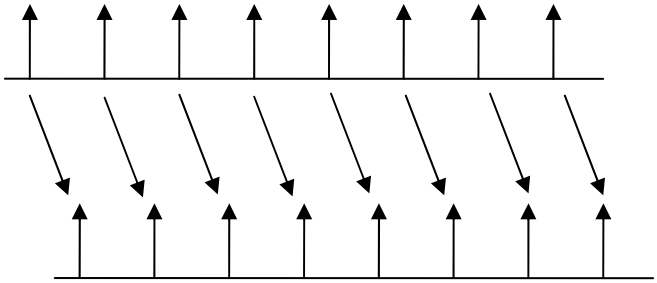
3) **Case-3:** If there is a frame skip at high-resolution encoder, the frame level information is retained and is consumed at next frame. (For example, if `SceneChange` occurs at Nth frame and it is frame skip, then the IDR frame is inserted at high-resolution at (N+1)th frame.) This way information is preserved and utilized at high-resolution encoder.

4) **Case-4:** If the low-resolution encoder is running at low frame rate and high resolution encoder is running at high frame rate, the metadata is consumed by the corresponding frame at high-resolution encoder. If the corresponding frame is frame skip then metadata is used by subsequent frame. This process continues if the frame skip persist and the metadata is pass to next frame until the new metadata is received from low-resolution encoder.

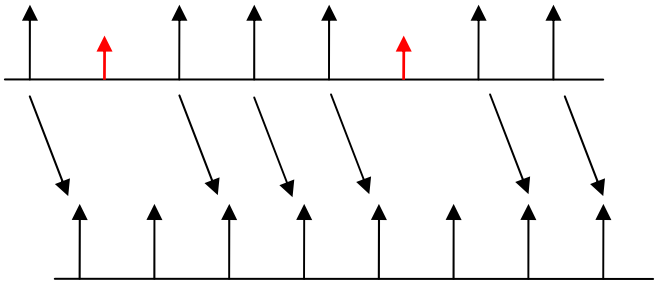
5) **Case-5:** If the low-resolution encoder is running at high frame rate and high-resolution encoder is running at low frame rate, the metadata information is retained inside the low-resolution codec until encoding at high reolution encoder starts and used accordingly at the high-resolution encoder.
   .

Frame Skip

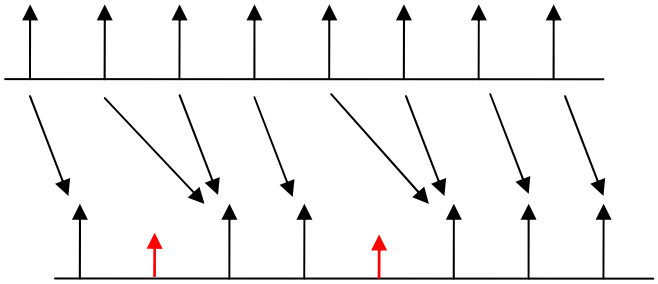
Case-1



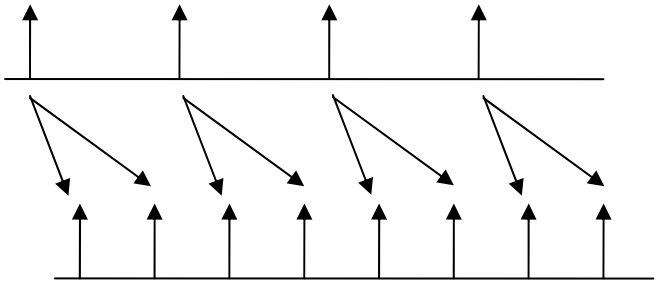
Case-2



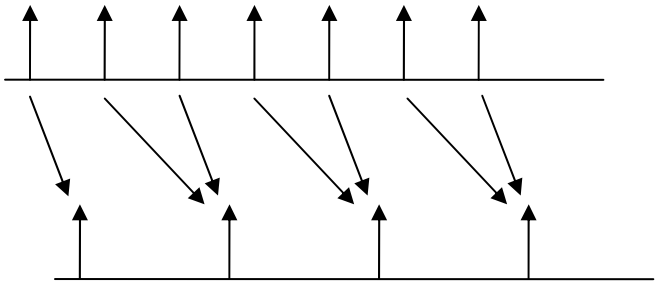
Case-3



Case-4



Case-5



Note:

Call `control()` function with the `XDM_SETPARAMS` command before starting encoding at low-resolution and high resolution instance for every frame (See section 3.1.3).

The following steps explains how to use STP feature of the DM365/DM368 based H264 encoder.

1) Set `metaDataGenerateConsume` flag value for low and high-resolution encoder to 1 and 2 respectively. If `meteDataGenerateConsume` is set to 0, no `metaData` is generated or consumed.

2) Request I/O buffers for two encoder instances as explained in the Chapter 3. In case of STP extra buffers are requested to store metadata. This is taken care inside the codec if the `metaDataGenerateConsume` flag is set appropriately.

3) After creating instances of both the encoders, initiate coding of low-resolution encoder.

4) Update metadata values in `frame_info` structure at low-resolution encoder. Once encoding operation is completed, copy the metadata into the output buffers of low-resolution encoder requested in the step 2.

5) If the frame skip occurs at low resolution, no metadata information is passed to the high-resolution encoder.

6) Now set the value of `metaDataGenerateConsume` flag for low-resolution encoder instance to 3, which means the metadata is generated but not yet consumed.

7) Before starting high resolution encoding, if `metaDataGenerateConsume` flag for low-resolution encoder instance is 3, copy metadata information from the output buffers of low-resolution encoder to the input buffer of the high-resolution encoder requested in the step 2. The metadata information from this input buffers is utilized appropriately by the codec at high-resolution encoder.

8) If the frame skip occurs at high-resolution encoder, metadata information will used by next frame appropriately.

9) Once the metadata is consumed and encoding is completed at high resolution encoder, set the value of `metaDataGenerateConsume` flag for low-resolution encoder instance to 1.

E.1 Example Usage:

In order to provide flexibility to generate/consume `metaData` information an extended dynamic parameter `metaDataGenerateConsume` is provided. Initially, it can take only three values: 0 (no `metaData` generated or consumed), 1 (Generate `metaData`) and 2 (Consume `metaData`). In case of low resolution encoder, we will set `metaDataGenerateConsume` to 1 and in case of high resolution encoder `metaDataGenerateConsume` is set to 2.

Example settings for low resolution encoder

In this case, the application requests for buffers which are used to pass frame level information from low resolution encoder to the high resolution encoder.

```
Set meetaDataGenerateConsume = 1

// generate metadata in low resolution encoder.

Output Buffer requirement by low resolution encoder
```

Structure Name	Buffer Size
FrameInfo Interface	sizeof(FrameInfo Interface);
MBInfo	(uiSize >> 4) * 4;
MBRowInfo	uiExtHeight * 4;

Where `uiSize` is the maximum number of pixels in a frame and `uiExtHeight` is the height of the frame in pixels.

Addresses of these buffers are passed to the codec where variables of the structures (See Section 4.6) are updated if `metaDataGenerateConsume` is set to 1. Once the encoding is completed, set the value of `metaDataGenerateConsume` flag for low-resolution instance to 3.

In current implementation, `MBinfo` and `MBRowinfo` structures are not populated, hence no buffers are requested.

E.1.1 Example settings for high resolution encoder

In this case, the application copies the metadata information from the buffers of low-resolution encoder to the buffers of high-resolution encoder. The high resolution encoder makes use of frame level information as and when it is required.

```
At application:

**********************************************************

If(metaDataGenerateConsume of low resolution is  3)

{

      If(metaDataGenerateConsume of high resolution is 2)
```

```
            Copy metadata from low-resolution to high
resolution;

}

************************************************************

Input Buffer requirement by high resolution encoder
```

Structure Name	Buffer Size
FrameInfo Interface	sizeof(FrameInfo Interface);

```
Inside Codec:

************************************************************

If(metaDataGenerateConsume of high resolution is 2) /*
Will be used in High resolution encoder */

{

            /*

      * Use metadata given by low resolution encoder to

      * take appropriate decisions.

            */

}

************************************************************
```

Once the information is consumed and encoding of high-resolution encoder is done, set the value of `metaDataGenerateConsume` flag of low resolution encoder instance to 1.

```
At application:

************************************************************

Set metaDataGenerateConsume flag of low resolution encoder
to 1;

************************************************************
```

Note:

- When setting `dynamicparams.metaDataGenerateConsume` = 2, for the high resolution encoder, the low resolution encoder must be run with `dynamic params.metaDataGenerateConsume` = 1, else severe quality degradation will occur.
- The usage of the generated `metaData` by the high resolution encoder is internal to the codec and no further input is required from the end user.
- If the current frame at low resolution encoder is encoded as IDR/I frame then no scene change information is passed to high resolution encoder.
- STP works with ROI enabled also.
- When the scenechange is detected at low resolution encoder, IDR frame is forced at high resolution encoder at the corresponding frame number.

This page is intentionally left blank

Appendix F

Revision History

This revision history highlights the changes made to the SPRUEU9A codec specific user guide to make it SPRUEU9B.

Table F-1.Revision History for H.264 Base/Main/High Profile Encoder on DM365/DM368

Section	Changes
Global	❑ There are no major changes in the user guide for this release of H.264 Base/Main/High Profile Encoder on DM365/DM368.