MPEG4 HDVICP Simple Profile Encoder on DM365

User's Guide

Literature Number: SPRUGR4A
September 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) MPEG4 HDVICP Simple Profile Encoder implementation on the DM365 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 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 – Revision History,** highlights the changes made to SPRUGR4 codec specific user guide to make it SPRUGR4A.

Related Documentation From Texas Instruments

The following documents describe TI's DSP algorithm standards such as, XDAIS and XDM. To obtain a copy of any of these TI documents, visit the Texas Instruments website at www.ti.com.

- *TMS320 DSP Algorithm Standard Rules and Guidelines* (literature number SPRU352) defines a set of requirements for DSP algorithms that, if followed, allow system integrators to quickly assemble production-quality systems from one or more such algorithms.
- *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360) describes all the APIs that are defined by the TMS320 DSP Algorithm Interoperability Standard (also known as XDAIS) specification.

Related Documentation

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

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

Abbreviations

The following abbreviations are used in this document.

Table 1-1. List of Abbreviations

Abbreviation	Description
AC	Alternate Current
AIR	Adaptive Intra Refresh
BIOS	TI's simple RTOS for DSPs
CBR	Constant Bit rate
D1	720x480 or 720x576 Resolutions in Progressive Scan
DC	Direct Current
DCT	Discrete Cosine Transform
DP	Data Partitioning
DM	Digital Media
DMA	Direct Memory Access
DSP	Digital Signal Processor
HDVICP	High Definition Video and Imaging Co-Processor sub-system

Abbreviation	Description
HEC	Header Extension Code
IDCT	Inverse Discrete Cosine Transform
MB	Macro Block
ME	Motion Estimation
MPEG	Moving Pictures Expert Group
MV	Motion Vector
PDM	Parallel Debug Manager
RTOS	Real Time Operating System
RVLC	Reversible Variable Length Coding
UMV	Unrestricted Motion Vector
VBR	Variable Bit rate
VGA	Video Graphics Array
VLC	Variable Length Coding
VOP	Video Object Plane (that is, Frame)
XDM	eXpressDSP Digital Media
YUV	Color space in luminance and chrominance form

Text Conventions

The following conventions are used in this document:

- Text inside back-quotes (‘) represents pseudo-code.
- Program source code, function and macro names, parameters, and command line commands are shown in a `mono-spaced` font.

Product Support

When contacting TI for support on this codec, quote the product name (MPEG4 HDVICP Simple Profile Encoder on DM365) and version number. The version number of the codec is included in the Title of the Release Notes that accompanies this codec.

Trademarks

Contents

Figures

This page is intentionally left blank

Tables

This page is intentionally left blank

Chapter 1

Introduction

This chapter provides a brief introduction to XDAIS, XDM, and DM365 software architecture. It also provides an overview of TI's implementation of the MPEG4 HDVICP Simple Profile Encoder on the DM365 platform and its supported features.

Topic **Page**

1.1 Software Architecture

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

DM365 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.

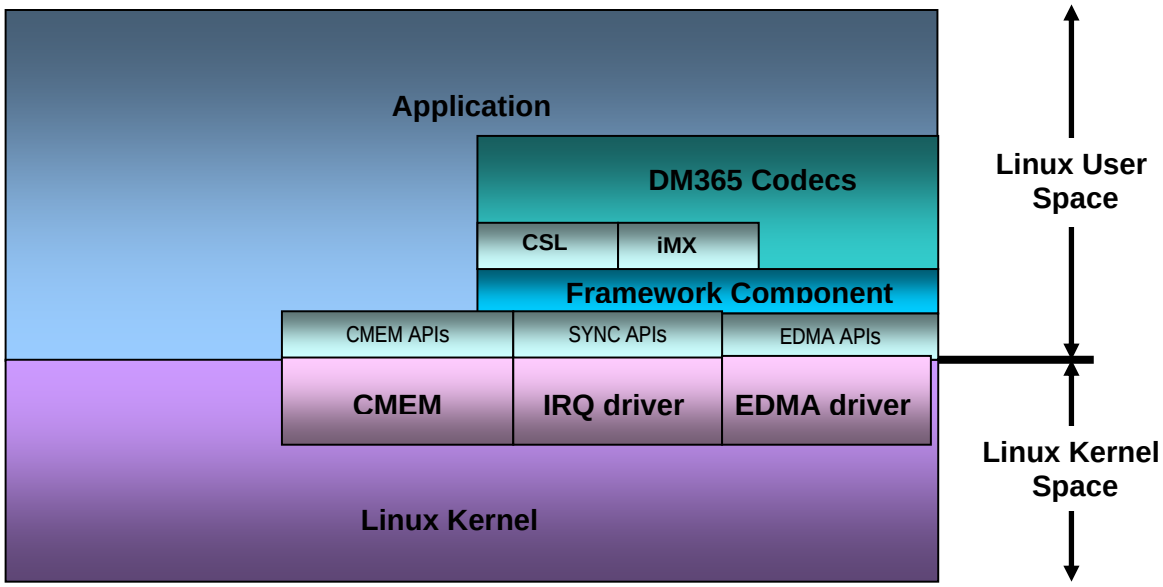


Figure 1-1. Software Architecture.

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:

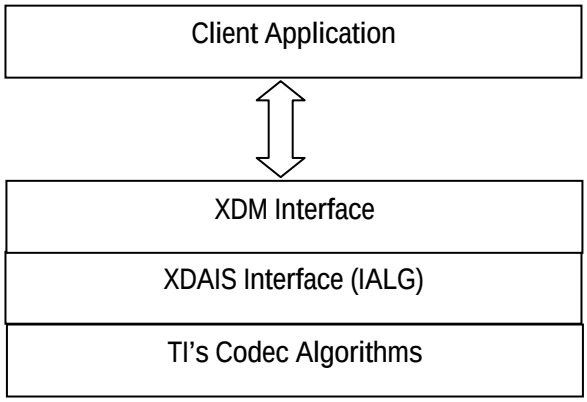
- `control()`
- `process()`

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, 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.

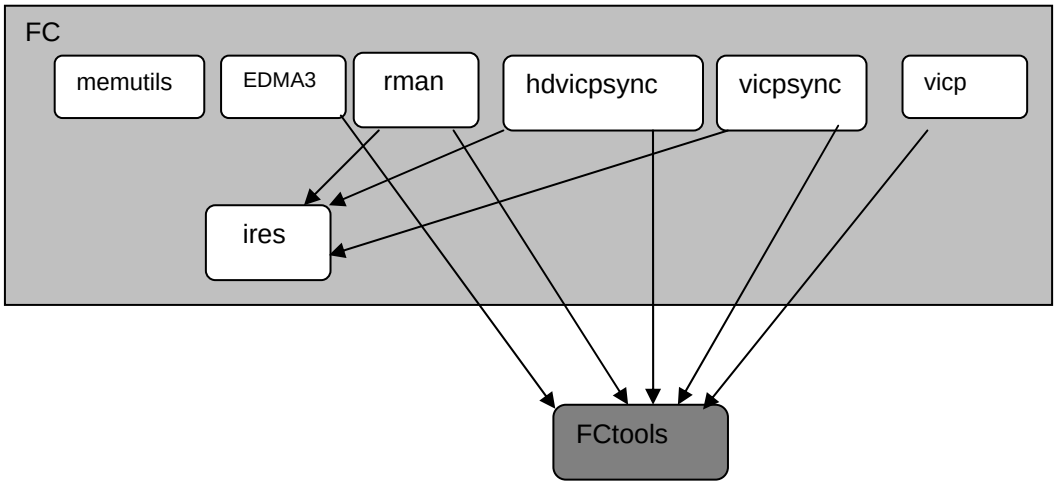


Figure 1-2. Framework Component Interfacing Structure.

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).

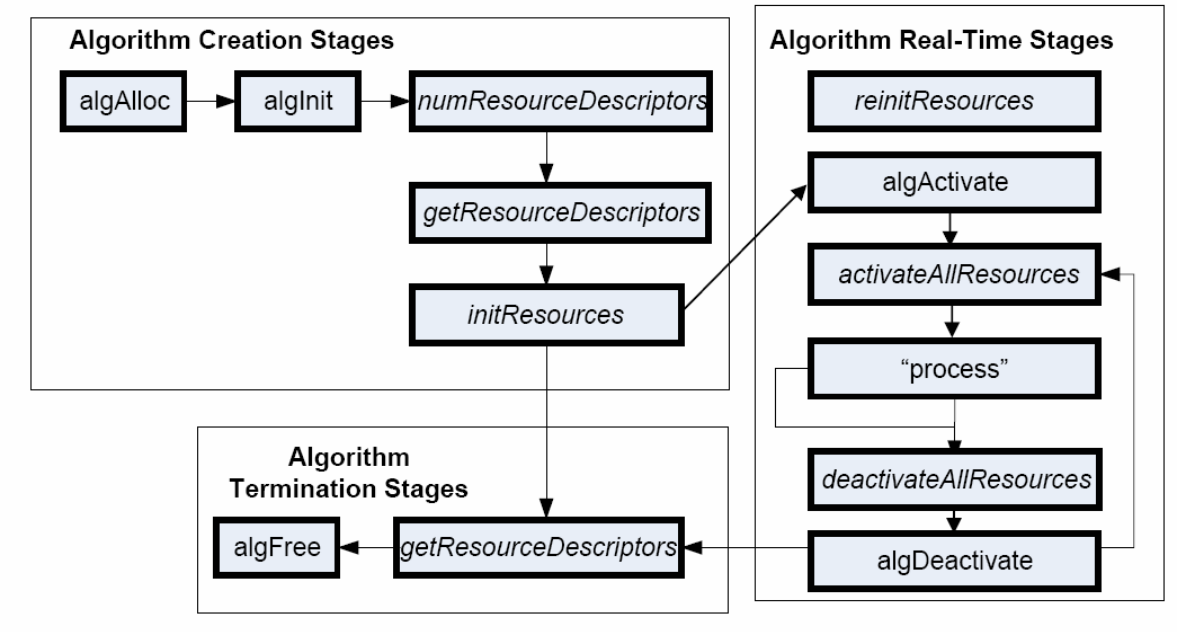


Figure 1-3. IRES Interface Definition and Function-calling Sequence.

In DM365, 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

1.3 Overview of MPEG4 HDVICP Simple Profile Encoder

MPEG4 (from ITU-T, also called as MPEG4/AVC) is a popular video coding algorithm enabling high quality multimedia services on a limited bandwidth network. MPEG4 standard defines several profiles and levels that specify restrictions on the bit stream and hence limits the capabilities needed to 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 MPEG4 profiles and their special features are:

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

Figure 1-4 depicts the working of the encoder.

The input video sequence for MPEG4 Encoder consists of frames, and accepts the input frames in YUV format. Video coding aims at providing a compact representation of the information in the video frames by removing spatial redundancies that exist within the frames and temporal redundancies that exist between successive frames. The MPEG4 standard is based on using the Discrete Cosine Transform (DCT) to remove spatial redundancies. Motion estimation and compensation is used to remove temporal redundancies.

All frames in a video sequence are categorized as I-frames and P-frames. I-frames called as intra-frames are encoded without reference to any other frame in the sequence, same as a still image would be encoded. In contrast, P-frames called as predicted frames or inter-frames depend on information from a previous frame for its encoding. The video frames that are close in time are similar. When encoding a video frame, you can use the information presented in a previously encoded frame.

One approach to achieve this goal is to consider the difference between the current frame and a previous reference frame, and encode the difference or residual. When two frames are very similar, the difference is more efficient to encode than encoding the original frame.

A more sophisticated approach to increase coding efficiency is to work at the macro block level in the current frame, instead of processing the whole frame all at once. This process is called motion compensation, or more precisely, motion compensated prediction. This is based on the assumption that most of the motion that the macro blocks undergo between frames is a translational motion.

Quantization is a significant source of compression in the encoder bit-stream. The basic idea of quantization is to eliminate as many of the non-zero DCT co-efficients corresponding to high frequency components. The quantized co-efficients are then rounded to the nearest integer value. The net effect of the quantization is usually a reduced variance between the original DCT co-efficients as compared to the variance between the original DCT co-efficients.

The reference frames in the encoder produces the output bit-stream in the compressed format as a sequence of data bits. With the help of a display driver, these bits are decoded and the output image can be seen on a display device such as a TV.

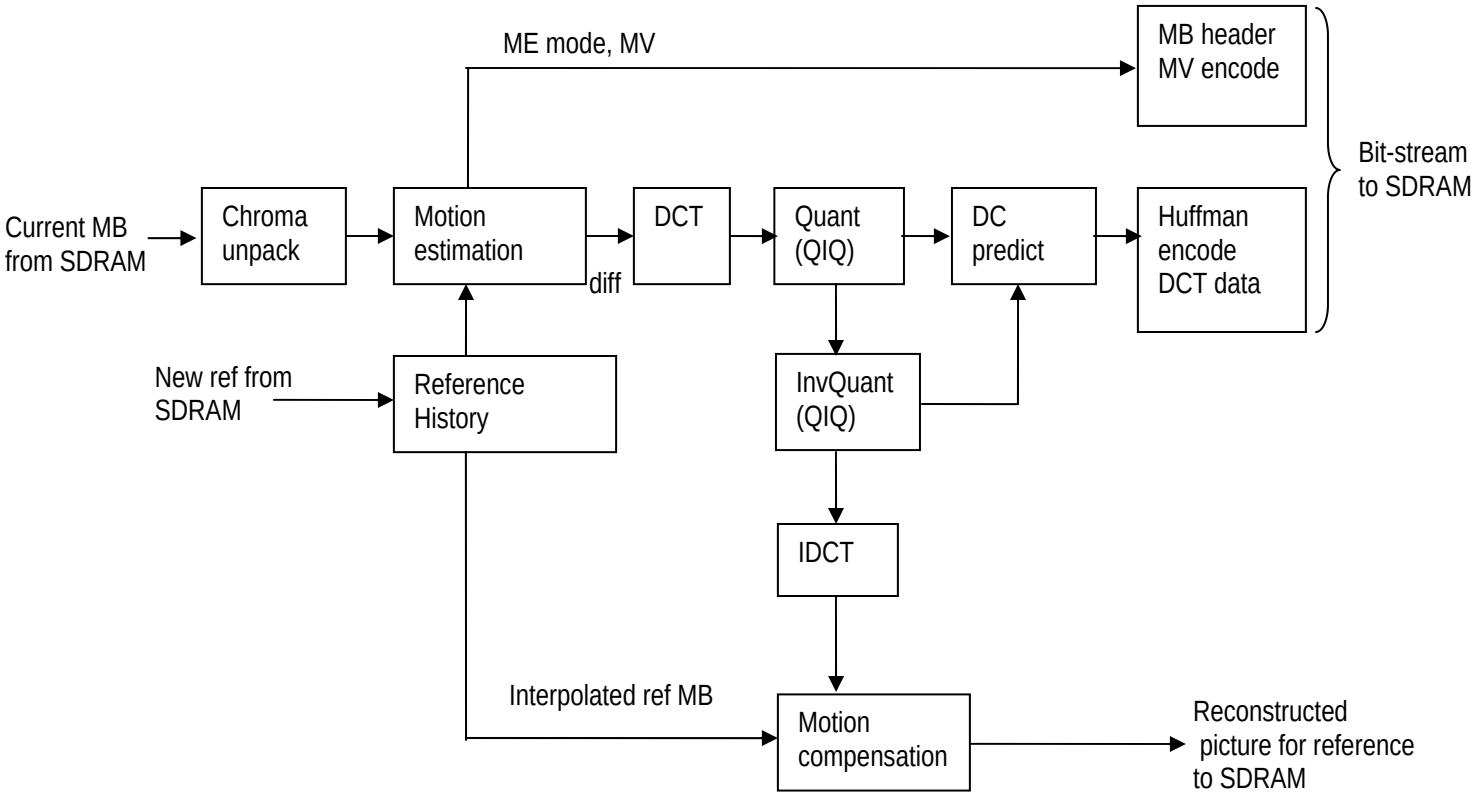


Figure 1-4. Block Diagram of MPEG-4 Encoder

From this point onwards, all references to MPEG4 Encoder mean MPEG4 High Profile (HP) Encoder only.

1.4 Supported Services and Features

This user guide accompanies TI's implementation of MPEG4 Encoder on the DM365 platform.

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

- ❑ eXpressDSP Digital Media (XDM 1.0 IVIDENC1) compliant
- ❑ Generates bit-streams compliant with the MPEG-4 standard
- ❑ Supports progressive frame type picture encoding
- ❑ Supports MPEG-4 Simple Profile levels 0, 0b, 1, 2, 3, 4A, and 5
- ❑ Supports H.263 Baseline Profile levels 10, 20, 30, 40 and 45.
- ❑ AC prediction supported up to CIF resolution
- ❑ TI's proprietary motion estimation supported

- Half Pixel Interpolation (HPI) for motion estimation supported
- 4MV (four motion vectors) mode is supported up to CIF resolution
- Supports Unrestricted Motion Vectors (UMV)
- Supports 2 types of ME search algorithms (Normal and Low power)
- Data Partitioning (DP) and Reversible Variable Length Coding (RVLC) supported up to CIF resolution
- Supports Adaptive Intra Refresh (AIR) up to D1 resolution
- Supports packet mode encoding up to CIF resolution
- Supports two modes for encoder quality (Encquality_mode). Standard quality and high quality.
- Supports resolutions up to Full HD (1920 x 1088) in MPEG4 mode and H.263 mode supports only standard resolutions like sub-QCIF (128x96), QCIF and CIF.
- Supports setting of Quantization Parameter (QP) for I-frames and P-frames
- Supports Rate Control (CBR and VBR)
- Supports Standard quality and high quality encoding.
- Supports YUV420 format for input frames

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 EVM (Set all the bits of SW4 and SW5 to low(0)
- 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_mpeg4enc_hdvicp_xx_xx_xx_xx_production.

Figure 2-5. Component Directory Structure for Linux.

Table 2-2 provides a description of the sub-directories created in the dm365_mpeg4enc_hdvicp_xx_xx_xx_xx_production 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_mpeg4enc_hdvicp_02_00_01_00_production.

Table 2-2. Component Directories for Linux.

Sub-Directory	Description
\package	Contains files related while building the package
\packages\ti\sdo\codecs\mpeg4enc_hdvicp\lib	Contains the codec library files on host

Sub-Directory	Description
\packages\ti\sdo\codecs\mpeg4enc_hdvicp\docs	Contains user guide and release notes
\packages\ti\sdo\codecs\mpeg4enc_hdvicp\apps\client\build\arm926	Contains the makefile to built sample test application
\packages\ti\sdo\codecs\mpeg4enc_hdvicp\apps\client\build\arm926\cmd	Contains a template (.xdt) file to used to generate linker command file
\packages\ti\sdo\codecs\mpeg4enc_hdvicp\apps\client\build\arm926\map	Contains the memory map generated on compilation of the code
\packages\ti\sdo\codecs\mpeg4enc_hdvicp\apps\client\test\src	Contains application C files
\packages\ti\sdo\codecs\mpeg4enc_hdvicp\apps\client\test\inc	Contains header files needed for the application code
\packages\ti\sdo\codecs\mpeg4enc_hdvicp\apps\client\test\testvecs\input	Contains input test vectors
\packages\ti\sdo\codecs\mpeg4enc_hdvicp\apps\client\test\testvecs\output	Contains output generated by the codec
\packages\ti\sdo\codecs\mpeg4enc_hdvicp\apps\client\test\testvecs\reference	Contains read-only reference output to be used for verifying against codec output
\packages\ti\sdo\codecs\mpeg4enc_hdvicp\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 dma_ti_dm365.a exists in the packages\ti\sdo\codecs\mpeg4venc_hdvicp\lib.

2) Verify that codec object library library mpeg4venc_ti_arm926.a exists in
the \packages\ti\sdo\codecs\mpeg4venc_hdvicp\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\mpeg4venc_hdvicp\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\mpeg4venc_hdvicp\client\build\arm926 and type the

command make. This generates an executable file mpeg4venc-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 ./mpeg4venc-r in the command window to run.

2.4 Configuration Files

This codec is shipped along with:

- Generic configuration file (testvecs_linux.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_linux.cfg for determining the input and reference files for running the codec and checking for compliance. The testvecs_linux.cfg file is available in the

\packages\ti\sdo\codecs\mpeg4enc_hdvicp\apps\client\test\testvecs\config sub-directory.

The format of the testvecs_linux.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_linux.cfg file is as shown:

```
0
../../test/testvecs/config/testparams.cfg
../../test/testvecs/input/colorful_toys_cif_5frms_420p.yuv
../../test/testvecs/output/colorful_toys_cif.m4v
../../test/testvecs/output/colorful_toys_cif_recon_420p.yuv
../../test/testvecs/config/colorful_toys_cif_meinfo.txt
```

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
####################################################
RateControlPreset= 5
EncoderPreset    = 3
ImageWidth       = 352      # Image width in Pels
ImageHeight      = 288      # Image height in Pels
FrameRate        = 30000    # Frames per second fps)*1000
BitRate          = 384000  # Bitrate(bps)
Level             = 5        # 0- Level0
                              1-Level1
                              2-Level2
                              3-Level3
                              4-Level4a
                              5-Level5
                              9-Level0b for mpeg4
                              # Level = 10, 20, 30, 40 and 45 for h263
ChromaFormat      = 9       # 9 => XMI_YUV_420P (only supported value)
RCAlgo            = 8       # 0 = IMP4HDVICPENC_RC_NONE ,
                             4 = IMP4HDVICPENC_RC_CBR,
                              8 = IMP4HDVICPENC_RC_VBR
maxDelay          = 1000    # Delay parameter fir Rate control in
                              milliseconds
aspectRatio        = 1      # 1=>1:1(Square),
                              2=>12:11,
                              3=>10:11,
                               4=>16:11,
                               5=> 40:33
pixelRange         = 1      # 0 :Y-16 to 235,
                                   Cb/Cr-16 to 240;
                                   1 : Y- 0 to 255, Cb/Cr-0 to 255
timerResolution = 30000 # No of Ticks per second (1 to 65535)
MPEG4_mode      = 1     #Mpeg4 (1)or H.263 (0)
IntraPeriod     = 29    # Period of I-Frames
useDataPartition   = 0 # Data partioning,
                              0 = off , 1 = on
useRVLC         = 0    # RVLC, 0 = off,1 = on
PacketSize      = 0    # No.of bits per packet (or to put resync marker), 0
```

```
                                means no packets
Four_MV_mode    = 0    # 8x8 partition
                        allowed or not 0 = off, 1 = on
useVOS          = 0    # VOS header is coded(1) or not(0)
useGOV          = 0    #GOV header is coded(1) or not(0)
useGOBSync      = 0    # 0 – GOB header off, 1 - insert GOB headers
Intra_QP        = 10    # Intra Qp value-range {2,31}
Inter_QP        = 10    # Inter Qp value-range {2,31}
useHEC          = 0    # HEC, 0 = off , 1 = on
NumAIRMBs       = 0    # No. of AIR MBs per frame
QPInit          = 8    # Init Qp value - range {2, 31}
QPMax           = 31   # Max Qp value-range {2, 31}
QPMin           = 2    # Min Qp value-range {2, 31}
UMV              = 1    # Unrestricted mottion vector 0 = off, 1 = on
FramesToEncode = 5    # Number of frames to be encoded
ME_Type           = 1   # 0 = Normal Search, 1 = Low Power
PerceptualRC    = 0     # This field is a reserved field and zero is the
                          only supported value
mvSADout       = 0      # Flag to enable/disable exporting of ME output
Encquality_mode = 1     # 1 => Enable Standard quality
                          0 => Disable Standard quality
```

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.

2.5 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_linux.cfg available in the \client\test\testvecs\config sub-directory. For details on the format of the testvecs_linux.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.

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 MPEG4 Encoder library. The main test application files are mpeg4venc_ti_arm926testapp.c and mpeg4encoderapp.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.

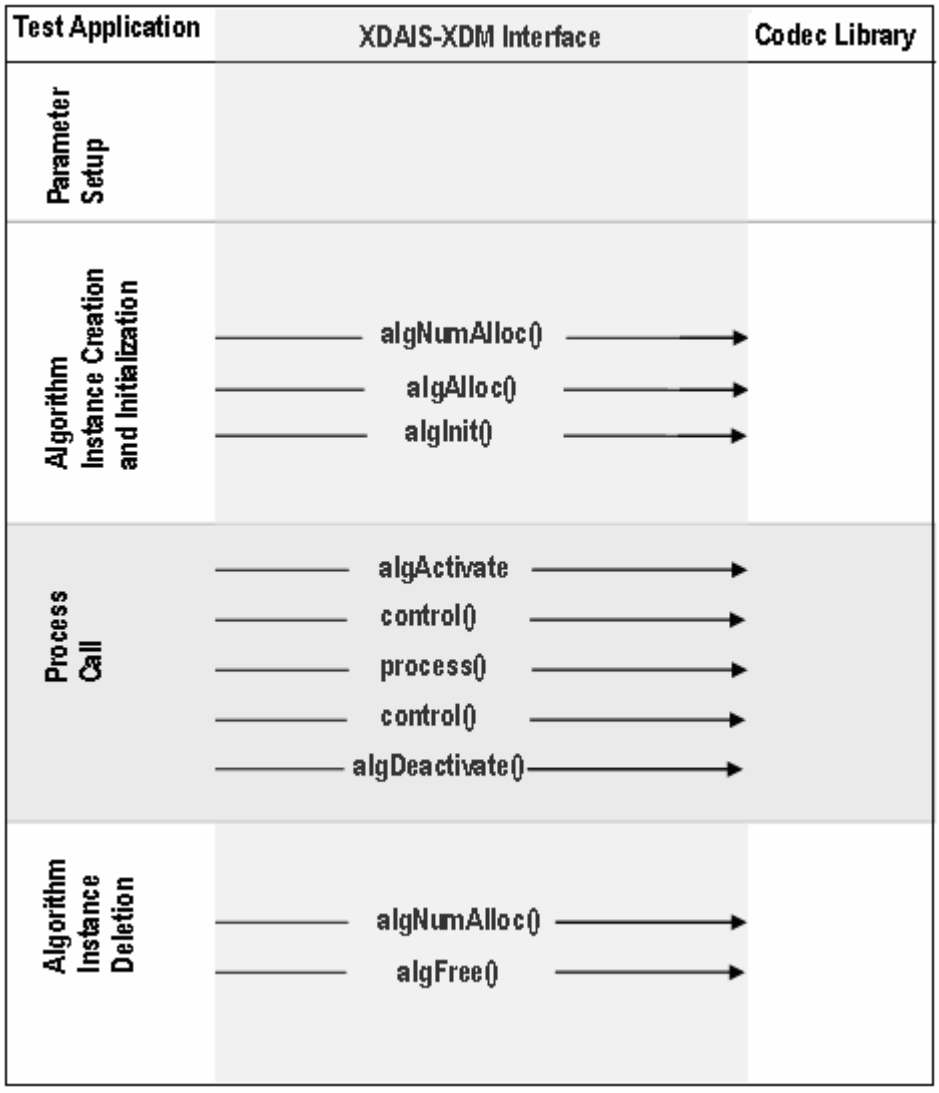


Figure 3-1. Test Application Sample Implementation

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_linux.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 `IMPEG4VENC_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 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`.

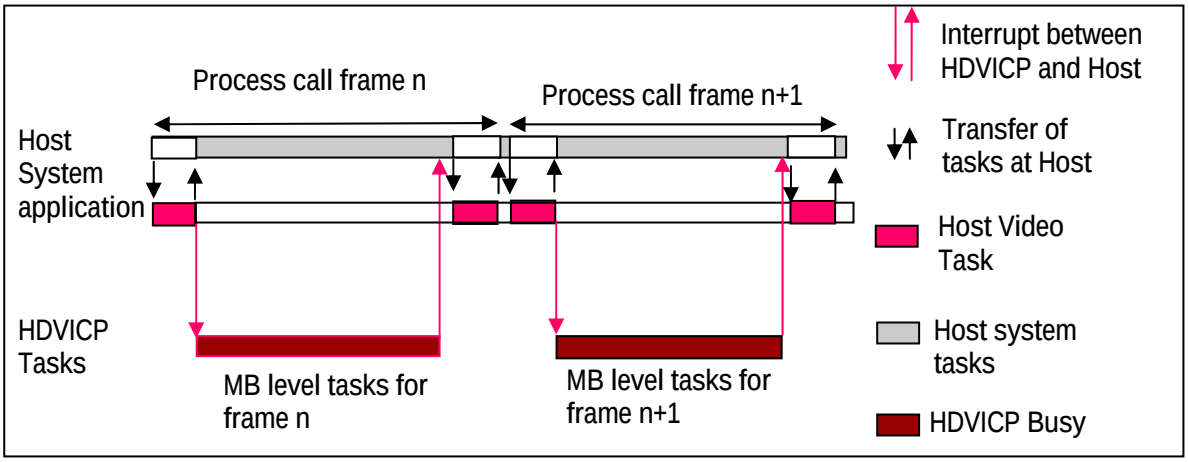


Figure 3-2. Process Call with Host Release

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

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

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

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

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

The while loop encapsulates frame level `process()` call and updates the input buffer pointer every time before the next call. The do-while loop breaks off either when an error condition occurs or when the input buffer exhausts.

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.

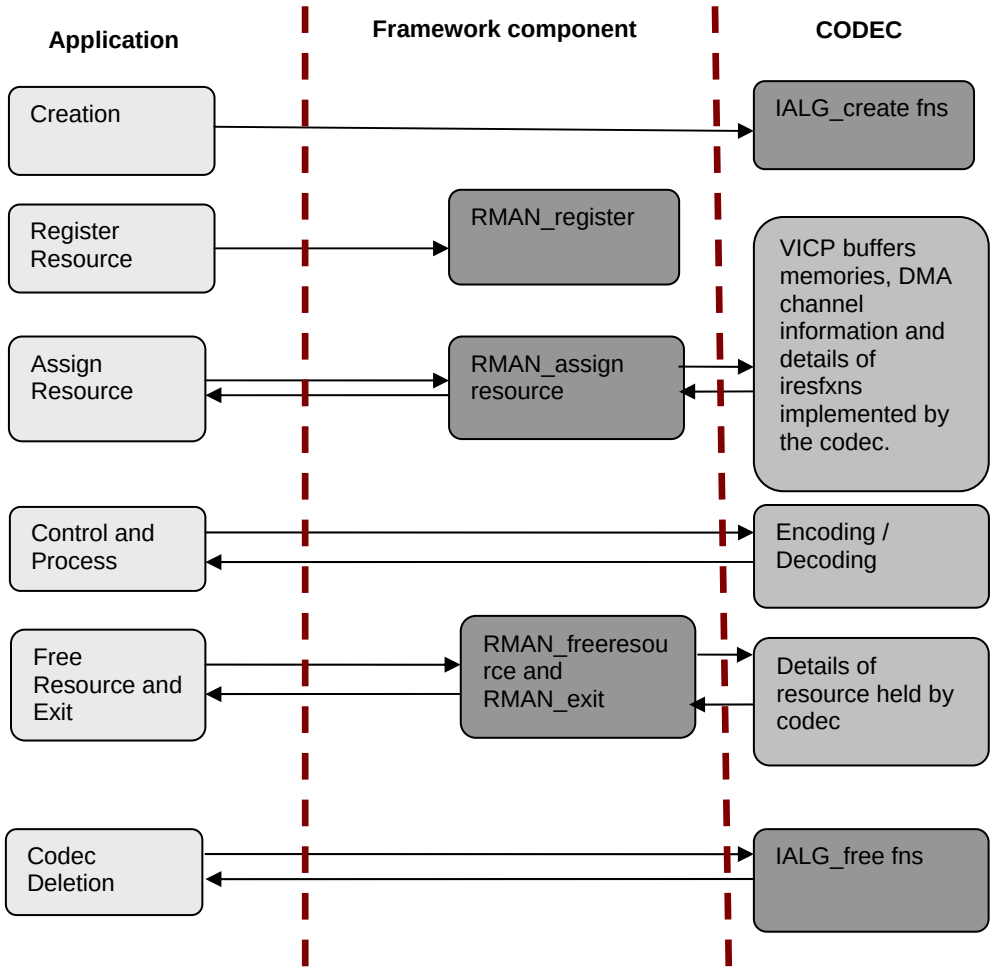


Figure 3-3. Resource Level Interaction.

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.

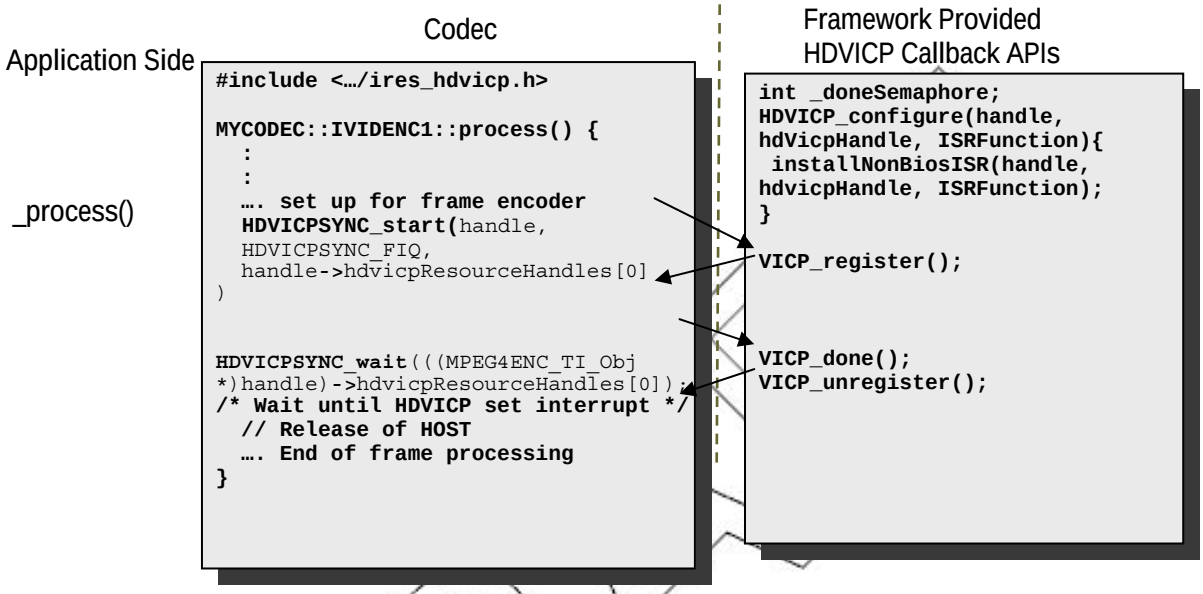


Figure 3-4. Interaction Between Application and Codec

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` 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.

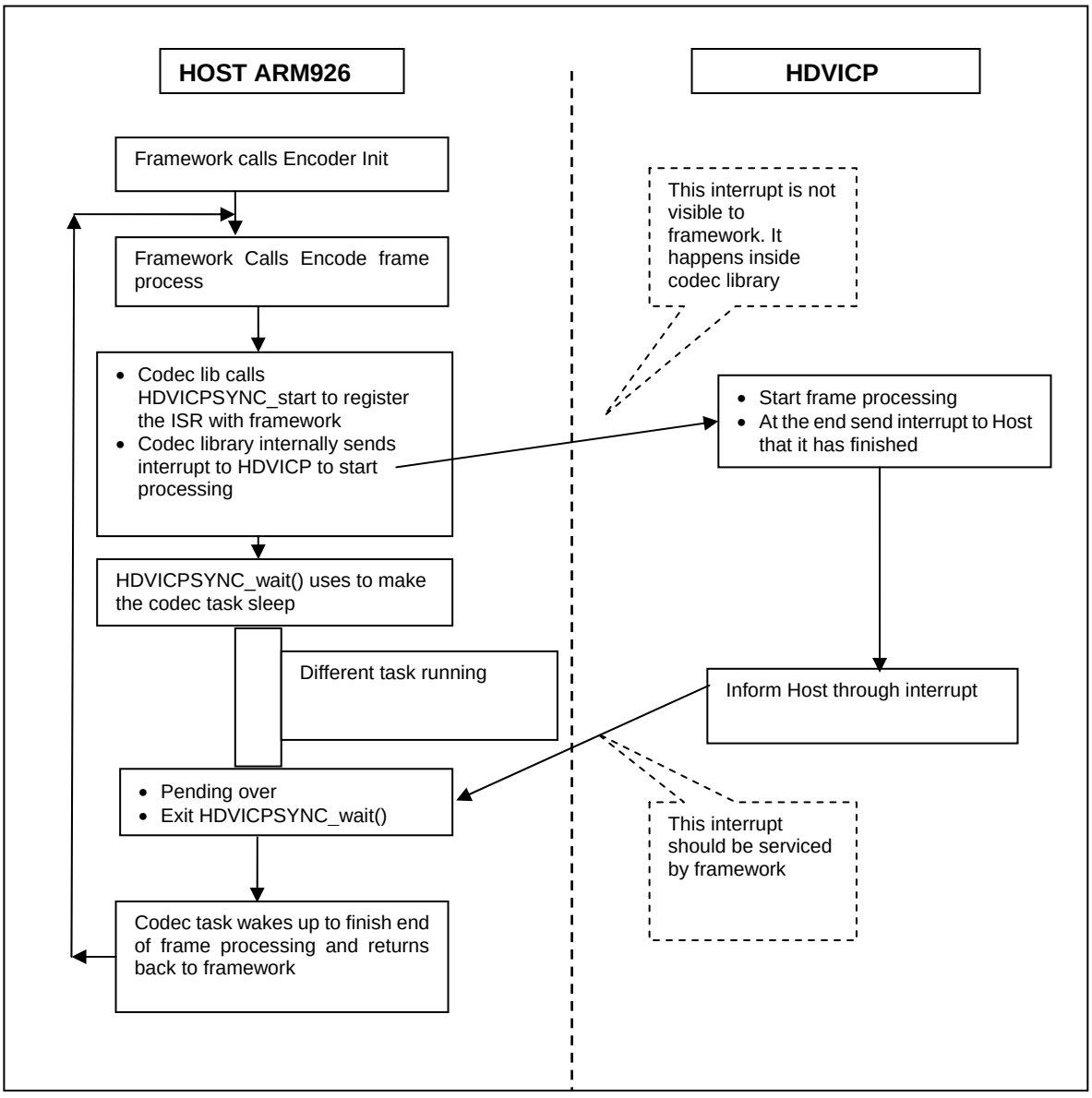


Figure 3-5. Interrupt Between Codec and Application.

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 mDDR (Mobile Double Data Rate).The host of DM365 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 util 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 MPEG4 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);
iresStatus = (IRES_Status)RMAN_register(&IRESMAN_HDVICP,
    (IRESMAN_Params *)&configParams);
iresStatus = RMAN_register(&IRESMAN_ADDRSPACE,
    (IRESMAN_Params *)&addrspaceConfigParams);

/*---------------- Encoder creation -----------------*/
handle = (IALG_Handle) MP4VENC_create();

/*Getting instance of algorithms that implements IALG and IRES
functions*/
iErrorFlag = RMAN_assignResources((IALG_Handle)handle,
                            &MPEG4VENC_TI_IRES,/* IRES_Fxns* */
                              1 /* scratchId */);

/* Get Buffer information */
MP4VENC_control (handle, XDM_GETBUFINFO);

/* Allocate memory for input and output frame buffers *//*SET
BASIC INPUT PARAMETERS */
MP4VENC_control (handle, XDM_SETPARAMS);

/* 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      */
/*-----------------------------------------------------*/
   retVal = MP4VENC_Encode
                  (
                    handle,
                    (XDM1_BufDesc *)&inputBufDesc,
                    (XDM_BufDesc *)&outputBufDesc,
                    (IVIDENC1_InArgs *)&inArgs,
                    (IVIDENC1_OutArgs *)&outArgs
                  );

/* Get the status of the encoder using control */
    MP4VENC_control (handle, XDM_GETSTATUS);
}

/* Free input and output frame buffers memory */
```

```
/* Free assigned resources */
RMAN_freeResources((IALG_Handle)(handle),
                   &MPEG4VENC_TI_IRES, /* IRES_Fxns* */
                  );

/* Delete the encoder Object handle*/
MP4VENC_delete (handle);

/* Unregister protocol*/
RMAN_unregister(&IRESMAN_EDMA3CHAN);
RMAN_unregister(&IRESMAN_HDVICP);
RMAN_unregister(&IRESMAN_ADDRSPACE);

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 MPEG4 Encoder.
	IVIDEO_IDR_FRAME	Intra coded frame that can be used for refreshing video content Not supported in this version of MPEG4 Encoder.
	IVIDEO_II_FRAME	Interlaced frame, both fields are I frames. Not supported in this version of MPEG4 Encoder.
	IVIDEO_IP_FRAME	Interlaced frame, first field is an I frame, second field is a P frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_IB_FRAME	Interlaced frame, first field is an I frame, second field is a B frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_PI_FRAME	Interlaced frame, first field is a P frame, second field is an I frame. Not supported in this version of MPEG4 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 MPEG4 Encoder.
	IVIDEO_BI_FRAME	Interlaced frame, first field is a B frame, second field is an I frame. Not supported in this version of MPEG4 Encoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_BP_FRAME	Interlaced frame, first field is a B frame, second field is a P frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_BB_FRAME	Interlaced frame, both fields are B frames. Not supported in this version of MPEG4 Encoder.
	IVIDEO_MBAFF_I_FRAME	Intra coded MBAFF frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_MBAFF_P_FRAME	Forward inter coded MBAFF frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_MBAFF_B_FRAME	Bi-directional inter coded MBAFF frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_MBAFF_IDR_FRAME	Intra coded MBAFF frame that can be used for refreshing video content. Not supported in this version of MPEG4 Encoder.
	IVIDEO_FRAMETYPE_DEFAULT	The default value is set to IVIDEO_I_FRAME.
IVIDEO_OutputFrameStatus	IVIDEO_FRAME_NOERROR	The output buffer is available.
	IVIDEO_FRAME_NOTAVAILABLE	The codec does not have any output buffers.
	IVIDEO_FRAME_ERROR	The output buffer is available and corrupted.
	IVIDEO_OUTPUTFRAMESTATUS_DEFAULT	By default, it is set to IVIDEO_FRAME_NOERROR.
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. Not supported in this version of MPEG4 Encoder.
IVIDEO_RateControlPreset	IVIDEO_NONE	No rate control is used

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_LOW_DELAY	Constant Bit-Rate (CBR) control for video conferencing. This is the default value.
	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 MPEG4 Encoder.
	IVIDEO_RATECONTROLPRESET_DEFAULT	Set to IVIDEO_LOW_DELAY
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 MPEG4 Encoder.
	XDM_LE_32	32-bit little endian stream. Not supported in this version of MPEG4 Encoder.
XDM_ChromaFormat	XDM_CHROMA_NA	Chroma format not applicable. Encoder assumes XDM_YUV_420SP
	XDM_YUV_420P	YUV 4:2:0 planar. Not supported in this version of MPEG4 Encoder.
	XDM_YUV_422P	YUV 4:2:2 planar. Not supported in this version of MPEG4 Encoder.
	XDM_YUV_422IBE	YUV 4:2:2 interleaved (big endian). Not supported in this version of MPEG4 Encoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_YUV_422ILE	YUV 4:2:2 interleaved (little endian). Not supported in this version of MPEG4 Encoder.
	XDM_YUV_444P	YUV 4:4:4 planar. Not supported in this version of MPEG4 Encoder.
	XDM_YUV_411P	YUV 4:1:1 planar. Not supported in this version of MPEG4 Encoder.
	XDM_GRAY	Gray format. Not supported in this version of MPEG4 Encoder.
	XDM_RGB	RGB color format. Not supported in this version of MPEG4 Encoder.
	XDM_YUV_420SP	YUV 420 semiplanar (Luma 1st plane, * CbCr interleaved 2nd plane)
	XDM_ARGB8888	Alpha plane Not supported in this version of MPEG4 Encoder
	XDM_RGB555	RGB 555 color format Not supported in this version of MPEG4 Encoder
	XDM_RGB565	RGB 556 color format Not supported in this version of MPEG4 Encoder
	XDM_YUV_444ILE	YUV 4:4:4 interleaved (little endian) Not supported in this version of MPEG4 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

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_FLUSH	Handle end of stream conditions. This command forces algorithm instance to output data without additional input. Not supported in this version of MPEG4 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).
	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

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	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:

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

4.1.2 MPEG4 Encoder Symbolic Constants and Enumerated Data Types

This section includes the following MPEG4 Encoder specific symbolic constants specified as either #define macros and/or enumerated C data types.

Table 4-2. List of Symbolic Constants.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IMP4HDVICPENC_RCAlgo	IMP4HDVICPENC_RC_NONE	No rate control is used
	IMP4HDVICPENC_RC_CBR	Constant Bit-Rate (CBR) control for video conferencing.
	IMP4HDVICPENC_RC_VBR	Variable Bit-Rate (VBR) control for local storage (DVD) recording. This is the default value.
	IMP4HDVICPENC_RC_DEFAULT	Set to IMP4HDVICPENC_RC_VBR
IMP4HDVICPENC_ChromaFormat	IMP4HDVICPENC_YUV_420IUV	YUV420 format with UV interleaved. This is the only supported chroma format for this version of encoder.
IMP4HDVICPENC_AspectRatio	IMP4HDVICPENC_AR_SQUARE	1:1 Square See Table 6-14 in MPEG-4 visual standard.
	IMP4HDVICPENC_AR_12_11	12:11 (625 type for 4:3 picture)

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
		See Table 6-14 in MPEG-4 visual standard.
	IMP4HDVICPENC_AR_10_11	10:11 (525 type for 4:3 picture) See Table 6-14 in MPEG-4 visual standard.
	IMP4HDVICPENC_AR_16_11	16:11 (625 type stretched for 16:9 picture) See Table 6-14 in MPEG-4 visual standard.
	IMP4HDVICPENC_AR_40_33	40:33 Square (525 type stretched for 16:9 picture) See Table 6-14 in MPEG-4 visual standard.
	IMP4HDVICPENC_AR_DEFAULT	Set to IMP4HDVICPENC_AR_SQUARE
IMP4HDVICPENC_PixelRange	IMP4HDVICPENC_PR_16_235	video_range=0, gives a range of Y from 16 to 235, Cb and Cr from 16 to 240. See Section 6.3.2 in MPEG-4 visual standard
	IMP4HDVICPENC_PR_0_255	video_range=1 gives a range of Y from 0 to 255, Cb and Cr from 0 to 255. See Section 6.3.2 in MPEG-4 visual standard.
	IMP4HDVICPENC_PR_DEFAULT	Set to IMP4HDVICPENC_PR_0_255
IMP4HDVICPENC_CmdId	IMP4HDVICPENC_PREPROCESS	Pre-processing operation to be performed on frame before encoding. Not supported in this version of MPEG4 Encoder.
IMP4HDVICPENC_Level	IMP4HDVICPENC_SP_LEVEL_0	MPEG-4 Simple profile level 0 Value = 0
	IMP4HDVICPENC_SP_LEVEL_0B	MPEG-4 Simple profile level 0b Value = 9
	IMP4HDVICPENC_SP_LEVEL_1	MPEG-4 Simple profile level 1 Value = 1
	IMP4HDVICPENC_SP_LEVEL_2	MPEG-4 Simple profile level 2 Value = 2
	IMP4HDVICPENC_SP_LEVEL_3	MPEG-4 Simple profile level 3 Value = 3
	IMP4HDVICPENC_SP_LEVEL_4A	MPEG-4 Simple profile level 4a Value = 4
	IMP4HDVICPENC_SP_LEVEL_5	MPEG-4 Simple profile level 5

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
		Value = 5
IMP4HDVICPENC_H263Level	IMP4HDVICPENC_H263_LEVEL_10	H263 baseline profile level 10
	IMP4HDVICPENC_H263_LEVEL_20	H263 baseline profile level 20
	IMP4HDVICPENC_H263_LEVEL_30	H263 baseline profile level 30
	IMP4HDVICPENC_H263_LEVEL_40	H263 baseline profile level 40
	IMP4HDVICPENC_H263_LEVEL_45	H263 baseline profile level 45
IMP4HDVICPENC_STATUS	MPEG4VENC_ERR_HANDLE_NULL	init/process/control/delete api called with NULL handle. Returns a fatal error
	MPEG4VENC_ERR_INCORRECT_HANDLE	init/process/control/delete api called with incorrect handle. Returns a fatal error.
	MPEG4VENC_ERR_MEMTAB_NULL	memtab null error in algInit/algFree. Returns a fatal error.
	MPEG4VENC_ERR_IVIDENC1_INITPARAMS_SIZE	videncParams.size shall be either sizeof(IVIDENC1_Params) or sizeof(IMPEG4VENC_Params). Otherwise a fatal error.
	MPEG4VENC_ERR_MEMTABS_SIZE	MemTabs size less than request size. Returns a fatal error.
	MPEG4VENC_ERR_MEMTABS_ATTRS	MemTabs attrs mismatch request value IALG_PERSIST/IALG_SCRATCH. Returns a fatal error.
	MPEG4VENC_ERR_MEMTABS_SPACE	MemTabs space mismatch request value IALG_EXTERNAL or IALG_DARAM0. Returns a fatal error.
	MPEG4VENC_ERR_MEMTABS_BASE_NULL	MemTab base pointers are NULL. Returns a fatal error.
	MPEG4VENC_ERR_MEMTABS_BASE_NOT_ALIGNED	MemTab base pointers not aligned to required sizes. Returns a fatal error

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	MPEG4VENC_ERR_MEMTABS_OVERLAP	MemTabs are overlapping. Returns a fatal error.
	MPEG4VENC_ERR_INV_CODEC_ID	Invalid Codec ID passed to the codec. Returns a fatal error
	MPEG4VENC_ERR_INPUTCHROMAFORMAT	inputChromaFormat will be IMP4HDVICPENC_YUV_420IUV/XDM_CHROMA_NA. Otherwise error returned by MPEG4VENC_TI_initObj.fatal and unsupported error
	MPEG4VENC_ERR_MAXFRAMERATE	0 < maxFrameRate <= 30000. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_MAXBITRATE	0 < maxBitRate <= 20000000. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_MAXWIDTH	MPEG4VENC_TI_MIN_WIDTH <= maxWidth <= MPEG4VENC_TI_MAX_WIDTH Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_MAXHEIGHT	MPEG4VENC_TI_MIN_HEIGHT<=maxHeight<=MPEG4VENC_TI_MAX_HEIGHT. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_MAX_TOTAL_MBS	number of MBs in frame exceeded limit. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_DATAENDIANNESS	dataEndianness shall be XDM_BYTE. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	MPEG4VENC_ERR_MAXINTERFRAMEINTERVAL	maxInterFrameInterval param not supported. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_RECONCHROMAFORMAT	reconChromaFormat shall be IMP4HDVICPENC_YUV_420IUV/XDM_CHROMA_NA. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_INPUTCONTENTTYPE	InputContentType to be only IVIDEO_PROGRESSIVE. Otherwise error returned by MPEG4VENC_TI_initObj.fatal and unsupported error 0 <= rateControlPreset <= IVIDEO_USER_DEFINED. Otherwise error returned by MPEG4VENC_TI_initObj. Fatal and unsupported error.
	MPEG4VENC_ERR_RATECONTROLPRESET	Mpeg4 (1) or H.263 (0) otherwise not supported. Otherwise error returned by MPEG4VENC_TI_initObj
	MPEG4VENC_ERR_MPEG4MODE	Mpeg4 mode can be either 0 or 1. ❑ 0 - H.263 mode ❑ 1 - Mpeg4 mode. Otherwise error returned by MPEG4VENC_TI_initObj. Fatal and unsupported error
	MPEG4VENC_ERR_LEVELIDC	levelIdc will be as in IMPEG4VENC_Level enums. Otherwise MPEG4VENC_TI_initObj returns error. Fatal and unsupported error
	MPEG4VENC_ERR_ENCPARM_USE_VOS	VOS header is coded or not coded. ❑ 1 - Coded ❑ 0 - Not coded Otherwise error returned by MPEG4VENC_TI_initObj . fatal and unsupported error
	MPEG4VENC_ERR_USE_GOV	GOV header is coded or not coded. ❑ 1 - Coded

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
		❑ 0 - Not coded. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_DATA_PART	Data portioning ❑ 0 - off ❑ 1 - on Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_RVLC	RVLC ❑ 0 - off ❑ 1 - on Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_DATA_PART_RVLC	If data partition is off then RVLC cannot be on. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_TIMER_RESOLUTION	Number of Ticks per second is between 1 and 65535. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_ASPECTRATIO	aspectRatio less than 1 or aspectRatio greater than 5 not supported. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_PIXELRANGE	pixelRange shall be 0 or 1. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_RESETHDVICPEVERYFRAME	resetHDVICPeveryFrame will be 0 or 1. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_METYPE	meType will be 0 or 1.If 0 normal search and if 1 low power. Otherwise error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_UNRESTRICTEDMV	unrestrictedMV will be 0 or 1. In H263 mode, Unrestricted MV is not

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
		supported. Error returned by MPEG4VENC_TI_initObj. fatal and unsupported error
	MPEG4VENC_ERR_IVIDENC1_PROCESS_ARGS_NULL	one or more of the arguments to process call is NULL. Error returned in ptOutArgs->extendedError by MPEG4VENC_TI_Encode call.fatal error
	MPEG4VENC_ERR_IVIDENC1_INARGS_SIZE	nArgs->size is not equal to either base params or extnd params. Error returned in ptOutArgs->extendedError by MPEG4VENC_TI_Encode call.fatal error
	MPEG4VENC_ERR_IVIDENC1_OUTARGS_SIZE	outArgs->size is not equal to either base params or extnd params. Error returned in ptOutArgs->extendedError by MPEG4VENC_TI_Encode call.fatal error
	MPEG4VENC_ERR_IVIDENC1_INARGS_INPUTID	inArgs->inputID is equal to 0. Error returned in ptOutArgs->extendedError by MPEG4VENC_TI_Encode call.fatal error
	MPEG4VENC_ERR_IVIDENC1_INARGS_TOPFIELDFIRSTFLAG	inArgs->topFieldFirstFlag is not set correctly. This is used only in case of interlace picture to set top field as the start. Error returned in ptOutArgs->extendedError by MPEG4VENC_TI_Encode call.fatal error
	MPEG4VENC_ERR_IVIDENC1_INBUFS	inBufs elements were not set correctly. Error returned in ptOutArgs->extendedError by MPEG4VENC_TI_Encode call.fatal error
	MPEG4VENC_ERR_IVIDENC1_INBUFS_BUFDESC	inBufs buffer descriptors were not set correctly. Error returned in ptOutArgs->extendedError by MPEG4VENC_TI_Encode call.fatal error
	MPEG4VENC_ERR_IVIDENC1_OUTBUFS	outBufs elements were not set correctly. Error returned in ptOutArgs->extendedError by MPEG4VENC_TI_Encode call.fatal

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
		error
	MPEG4VENC_ERR_IVIDENC1_OUTBUFS_NULL	outBufs buffers are NULL. Error returned in ptOutArgs->extendedError by MPEG4VENC_TI_Encode call.fatal error
	MPEG4VENC_ERR_CODEC_INACTIVE	Fatal error if process/control API is called without prior activate(). Error returned in ptOutArgs->extendedError by MPEG4VENC_TI_Encode call.fatal error
	MPEG4VENC_ERR_CODEC_NOT_INITIALIZED	Fatal error if process/control API is called without prior initializations().Error returned in ptOutArgs->extendedError by MPEG4VENC_TI_Encode call. fatal error
	MPEG4VENC_ERR_INPUTWIDTH_NON_MULT_OF_2	If input width is not a multiple of 2Not supported and fatal. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError
	MPEG4VENC_ERR_INPUTHEIGHT_NON_MULT_OF_2	If input height is not a multiple of 2 Not supported and fatal. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError
	MPEG4VENC_ERR_INPUTWIDTHHEIGHT	MPEG4VENC_TI_MIN_WIDTH <= inputWidth <= maxWidth; and MPEG4VENC_TI_MIN_HEIGHT <= inputWidth <= maxHeight; else fatal. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError. fatal and unsupported error
	MPEG4VENC_ERR_CAPTUREWIDTH	captureWidth should be 0 or >= inputWidth. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError. fatal and unsupported error

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	MPEG4VENC_ERR_GENERATEHEADER	(generateHeader = 0) or (generateHeader = 1) is only supported. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError. fatal and unsupported error
	MPEG4VENC_ERR_INTERFRAMEINTERVAL	interFrameInterval will be 0 or 1 Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError. fatal and unsupported error
	MPEG4VENC_ERR_BITRATE	(0 < BitRate <= maxBitRate). Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError. fatal and unsupported error
	MPEG4VENC_ERR_REFFRAMERATE_MISMATCH	refFrameRate is not equal to targetFrameRate then it is not supported. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError fatal and unsupported error
	MPEG4VENC_ERR_INTRAFRAMEINTERVAL	intraFrameInterval greater than 0 is supported. Indicates the number of frames between two successive I-frames. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError. fatal and unsupported error
	MPEG4VENC_ERR_TARGETFRAMERATE	(0 < targetFrameRate <= maxFrameRate). targetFrameRate will be multiple of 500. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError. fatal and unsupported error
	MPEG4VENC_ERR_FORCEFRAME	forceFrame other than

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
		IVIDEO_NA_FRAME or IVIDEO_I_FRAME or IVIDEO_P_FRAME.IVIDEO_IDR_FRAME not supported. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError. fatal and unsupported error
	MPEG4VENC_ERR_GOBHEADER	Number of gob headers cannot be more than number of MB rows in frame. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError. fatal and unsupported error
	MPEG4VENC_ERR_TIMESCALE	timeScale<0 or (timeScale*1000<targetFrameRate). Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error .
	MPEG4VENC_ERR_INTERFRAMEQP	QP value limitation from MPEG-4 standard - QP range 0 to 31. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error .
	MPEG4VENC_ERR_INTRAFRAMEQP	QP value limitation from MPEG-4 standard - QP range 0 to 31. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error .
	MPEG4VENC_ERR_INITQ	QP value limitation for MPEG-4 is 2 to 31 and QP value limitation for H263 is 8 to 31 In case of any value other than specified as above this error is

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
		returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error .
	MPEG4VENC_ERR_QPMAX	QP value limitation from MPEG-4 standard - QP range 0 to 31 Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error
	MPEG4VENC_ERR_QPMIN	QP value limitation from MPEG-4 standard - QP range 0 to 31 Maximum value is 31. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error Minimum value is 2. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error
	MPEG4VENC_ERR_PACKETSIZE	Packet size should be either 0 or more than1024 bits for this implementation. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error
	MPEG4VENC_ERR_AIRRATE	Number of intra refresh MBs per frame. Minimum value is 0 and maximum can be number of MB's in (frame-1). Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	MPEG4VENC_ERR_4MVMODE	Either 0 or 1 ❑ 0 – 4MV mode off ❑ 1-- 4 MV mode on Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error
	MPEG4VENC_ERR_HECMODE	Can take only two values. 0 or 1 . ❑ 0 – useHEC is off ❑ 1—useHEC is on0 Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error
	MPEG4VENC_ERR_RCMETHOD	Supports 0, 4 or 8 only. ❑ 0 - No Rate control ❑ 4 - CBR is on ❑ 8 -VBR is on Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error
	MPEG4VENC_ERR_MAXDELAY	Maximum delay should be more than 100. Otherwise this error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError . fatal and unsupported error
	MPEG4VENC_ERR_PERCEPTUALRC	This is a reserved error code
	MPEG4VENC_ERR_INVALID_H263_LEVEL	This is used in H.263 mode to indicate any invalid input resolution set for the particular level specified. fatal and unsupported error
	MPEG4VENC_ERR_MV_SAD_OUT_FLAG	mvSADoutFlag will be 0 or 1.Else not supported.
	MPEG4VENC_ERR_SP_LEVEL_RESOLUTION	Error code to indicate if a given resolution is not supported for the level specified. This error is

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
		returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError
	MPEG4VENC_ERR_SP_LEVEL_MAX_MBS_MAX_BITRATE	Error code if target bit rate exceeds that associated with a level or if maximum MB's per frame is greater than 99. This error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError
	MPEG4VENC_ERR_SP_LEVEL_MAX_ENCODED_MBS	Error code if maximum encoded MB's per second exceeds that associated with a level. This error is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError
	MPEG4VENC_WARN_SP_LEVEL_DP	Warning code if packet size exceeds 2048 for data partition case. This warning is returned by MPEG4VENC_TI_control call in status.videncStatus.extendedError

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`
- `IVIDEO1_BufDesc`
- `IVIDEO1_BufDescIn`
- `IVIDENC1_Fxns`
- `IVIDENC1_Params`
- `IVIDENC1_DynamicParams`
- `IVIDENC1_InArgs`
- `IVIDENC1_Status`
- `IVIDENC1_OutArgs`
- `IVIDENC1_MbData`

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 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.

4.2.1.3 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

4.2.1.4 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	Mask filled by the algorithm, declaring how the buffer was accessed by the algorithm processor.

4.2.1.5 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 MPEG4 HDVICP Simple 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 1920x1088in YUV420SP format are:

  Y buffer = 1920 * 1088
  UV buffer = 1920 * 544

The above input buffer size calculation is done assuming that the capture width is same as image width. For details on capture width, see Section 4.2.1.10.

- 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.

  This worst case buffer size is applicable only for natural video sequences. For synthetic or noise kind of sequences, application has to allocate higher size output buffer say 5 to 10 times input frame size.

4.2.1.6 IVIDEO1_BufDesc

Description

This structure defines the buffer descriptor for output reconstructed buffers.

Fields

Field	Datatype	Input/ Output	Description
`numBufs`	`XDAS_Int32`	Output	Number of buffers
`frameWidth`	`XDAS_Int32`	Output	Width of the video frame to be encoded
`frameHeight`	`XDAS_Int32`	Output	Height of the video frame to be encoded
`framePitch`	`XDAS_Int32`	Output	Frame pitch used to store the frame.
`bufDesc[IVIDEO_MAX_YUV_BUFFERS]`	`XDM1_SingleBufDesc`	Output	Pointer to the vector containing buffer addresses. `IVIDEO_MAX_YUV_BUFFERS` macro is defined as 3.
`extendedError`	`XDAS_Int32`	Output	Extended error information
`frameType`	`XDAS_Int32`	Output	Type of the video frame. This takes one of the values from data type `IVIDEO_FrameType` as described in Table 4-1.
`topFieldFirstFlag`	`XDAS_Int32`	Output	Flag to indicate when the application should display the top field first Note: This feature is not supported in this version of MPEG4 Encoder on DM365.
`repeatFirstFieldFlag`	`XDAS_Int32`	Output	Flag to indicate when the first field should be repeated Note: This feature is not supported in this version of MPEG4 Encoder on DM365.
`frameStatus`	`XDAS_Int32`	Output	Status of the output frame `IVIDEO_OutputFrameStatus`. Not supported in this version of MPEG4

Field	Datatype	Input/ Output	Description
			Encoder.
repeatFrame	XDAS_Int32	Output	Number of times the display process should repeat. Note: This feature is not supported in this version of MPEG4 Encoder on DM365.
contentType	XDAS_Int32	Output	The output data content type. Only supported value is IVIDEO_PROGRESSIVE.
chromaFormat	XDAS_Int32	Output	Input chroma format. Only supported value is XDM_YUV_420SP

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
frameHeight	XDAS_Int32	Input	Height of the video frame
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. 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 IMP4HDVICPENC_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. Only IVIDEO_LOW_DELAY and IVIDEO_STORAGE are supported in this version. Default value = IVIDEO_STORAGE.
maxHeight	XDAS_Int32	Input	Maximum video height to be supported in pixels. Maximum height supported in this version is 1088. Minimum height supported is 64. Default value is 1088. Note: Total number of macroblocks that can be encoded per frame is limited to 8160.
maxWidth	XDAS_Int32	Input	Maximum video width to be supported in pixels. Maximum width supported in this version is 1920. Minimum width supported is 64. Default value is 1920. Note: Total number of macroblocks that can be encoded per frame is limited to 8160.
maxFrameRate	XDAS_Int32	Input	Maximum frame rate in fps * 1000 to be supported. Default value = 30000.
maxBitRate	XDAS_Int32	Input	Maximum bit-rate to be supported in bits per second. Maximum bit-rate supported in this version is 20000000.

Field	Data type	Input/ Output	Description
dataEndianness	XDAS_Int32	Input	Endianness of input data. See XDM_DataFormat enumeration for details. Only XDM_BYTE is supported in this version. 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 IMP4HDVICPENC_ChromaFormat enumeration for details. Only IMP4HDVICPENC_YUV_420IUV is supported in this version. Default value is XDM_YUV_420SP
inputContentType	XDAS_Int32	Input	Input content type. See IVIDEO_ContentType enumeration for details. Only IVIDEO_PROGRESSIVE is supported in this version. Default value = IVIDEO_PROGRESSIVE.
reconChromaFormat	XDAS_Int32	Input	Chroma formats for the reconstruction buffers. XDM_CHROMA_NA is supported, that is, Recon format is same as input chroma format. Other values are not supported in this version. Default value is XDM_YUV_420SP.

Note:

- The maximum video height and width supported are 1088 and 1920 respectively.
- For the supported `maxBitRate` values, see Annex N in *ISO/IEC 14496-2*.

The following fields of `IVIDENC1_Params` data structure are level dependent:

- `maxHeight`
- `maxWidth`
- `maxFrameRate`
- `maxBitRate`

The `maxFrameRate` is calculated based on maximum MB/sec. See Annex N in *ISO/IEC 14496-2* for details.

For CIF, if maximum MB/sec is 11880, this implies `maxFrameRate` is 30 fps as CIF has 396 MB/frame.

Encoder works with non-standard compliant values as well as for the above parameters. The application should ensure that the parameters are set to correct values.

For `rateControlPreset`, `IVIDEO_TWOPASS` is not supported in this version of MPEG4 Encoder.

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 `IMP4HDVICPENC_DynamicParams` structure.

Field	Data type	Input/ Output	Description
inputHeight	XDAS_Int32	Input	Height of input frame in pixels. Input height can be changed before start of encoding with in the limits of Max height set in creation phase. inputHeight must be multiple of two. Minimum height supported is 64. Note: 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 shall set input height to actual width but should provide the padded input YUV data buffer to encoder. Default value is 1088.
inputWidth	XDAS_Int32	Input	Width of input frame in pixels. Input width can be changed before start of encoding with in the limits of Max width set in creation phase. inputWidth must be multiples of two. Minimum width supported is 64. Note: When the input width is a non-multiple of 16, the encoder expects the application to pad the input frame to the nearest multiple of 16 to the right of the frame. In this case, application shall set inputWidth to actual width but should provide the padded input YUV data buffer to encoder. Default value is 1920.
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 = 30000
targetFrameRate	XDAS_Int32	Input	Target frame rate in fps * 1000. For example, if the frame rate is 30, set this field to 30000. Target frame rate should be less than maximum frame rate (see IVIDENC1_Params) value provided in creation phase and should be greater than 0 fps. Note: For this version to work normally, both refFrameRate and targetFrameRate should be set to same value. Default value is 30*1000.

Field	Data type	Input/ Output	Description
targetBitRate	XDAS_Int32	Input	Target bit-rate in bits per second. For example, if the bit-rate is 2 Mbps, set this field to 2097152. Default value is 4000000bps. Target bit- rate should be less than maximum bit-rate (see IVIDENC1_Params) value provided in creation phase and should be greater than or equal to 0. When targetBitRate = 0, encoder assumes fixed quantization values passed with qpIntra and qpInter parameters (see IMP4HDVICPENC_DynamicParams) and no rate control algorithm will be used for encoding.
intraFrameInterval	XDAS_Int32	Input	Interval between two consecutive intra frames. ❑ 0 : I-frame followed by all P-frames ❑ 1 : No inter frames(all intra frames) ❑ n > 1 : (n-1) frames coded as p-frames between every two I-frames Default value is 15.
generateHeader	XDAS_Int32	Input	Encode entire access unit or only header. See XDM_EncMode enumeration for details. Default value = XDM_ENCODE_AU.
captureWidth	XDAS_Int32	Input	Capture width parameter enables the application to provide input buffers with different line width (pitch) alignment than image width. For progressive content, if the parameter is set to: ❑ 0 - Encoded image width is used as pitch. ❑ < encoded image width - If capture width is set less than encoded image width, then capture width is ignored and encoded image width is used as pitch. ❑ >= encoded image width - capture width is used as pitch. Default value = 0

Field	Data type	Input/ Output	Description
forceFrame	XDAS_Int32	Input	Force the current (immediate) frame to be encoded as a specific frame type. For example, this field will be: ❑ 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 (specific to H.264 codecs). Not supported for MPEG-4 ❑ IVIDEO_P_FRAME - Force the frame to be encoded as a P frame. Not supported in this version of MPEG-4 encoder ❑ IVIDEO_B_FRAME - Force the frame to be encoded as a B frame. Not supported in this version Default value is 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
- The value of the refFrameRate and targetFrameRate should be the same.
- targetBitRate <= maxBitRate
- The inputHeight and inputWidth must be multiples of two.

When Input height/width are non-multiple of 16, encoder expects application to pad the input frame to the nearest multiple of 16 at the bottom/right of the frame. In this case, application should set input Height/Width to actual height/width, but should provide the padded input YUV data buffer to encoder.

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. This is not used in this version.

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 in section 4.1.1and codec specific code in section 4.1.2 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 in section 4.1.1and codec specific code in section 4.1.2 for details.
bytesGenerated	XDAS_Int32	Output	The number of bytes generated after each process/encode call.
encodedFrameType	XDAS_Int32	Output	Frame types for video. See IVIDEO_FrameType enumeration for details. Only IVIDEO_I_FRAME, IVIDEO_P_FRAME are supported in this version.
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 with the encoded buffer. This is also used to free up 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 bitstream.
reconBufs	IVIDEO1_BufDesc	Output	Pointer to reconstruction buffer descriptor.

4.2.1.14 IVIDENC1_MbData

‖ Description

This structure defines the macro-block related encoding parameter estimates. An array of IVIDENC1_MbData is provided to the encoder when IVIDENC1_DynamicParams::mbDataFlag is set.

> **Note:**
>
> This structure/feature is not used/supported in the this version of MPEG-4 encoder implementation on DM365.

‖ Fields

Field	Datatype	Input/ Output	Description
mbMode	XDAS_Int32	Input	MB encoding mode
QP	XDAS_Int32	Input	Quantization Parameter estimate
mvFwdXY	XDAS_Int32	Input	Forward Motion vector estimate
mvBwdXY	XDAS_Int32	Input	Backward motion vector estimate

4.2.2 MPEG4 Encoder Data Structures

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

- IMP4HDVICPENC_Params
- IMP4HDVICPENC_DynamicParams
- IMP4HDVICPENC_exportMEdata
- IMP4HDVICPENC_InArgs
- IMP4HDVICPENC_Status
- IMP4HDVICPENC_OutArgs
- IMP4HDVICPENC_Fxns

4.2.2.1 IMP4HDVICPENC_Params

Description

This structure defines the creation parameters and any other implementation specific parameters for a MPEG4 Encoder instance object. The creation parameters are defined in the XDM data structure, IVIDENC1_Params.

Fields

Field	Datatype	Input/ Output	Description
videncParams	IVIDENC1_Params	Input	See IVIDENC1_Params data structure for details.
MPEG4_mode	XDAS_Int32	Input	For Mpeg4 (1) and H.263 (0) the default value is 1.
levelIdc	XDAS_Int32	Input	Profile level indication for MPEG4/H263. Following values are supported for MPEG-4 mode: ❑ IMP4HDVICPENC_SP_LEVEL_0 ❑ IMP4HDVICPENC_SP_LEVEL_0B ❑ IMP4HDVICPENC_SP_LEVEL_1 ❑ IMP4HDVICPENC_SP_LEVEL_2 ❑ IMP4HDVICPENC_SP_LEVEL_3 ❑ IMP4HDVICPENC_SP_LEVEL_4A ❑ IMP4HDVICPENC_SP_LEVEL_5 See IMP4HDVICPENC_Level enumeration for details. Default value is

Field	Datatype	Input/ Output	Description
			IMP4HDVICPENC_SP_LEVEL_5. Following values are supported for H263 mode: ❑ IMP4HDVICPENC_H263_LEVEL_10 ❑ IMP4HDVICPENC_H263_LEVEL_20 ❑ IMP4HDVICPENC_H263_LEVEL_30 ❑ IMP4HDVICPENC_H263_LEVEL_40 ❑ IMP4HDVICPENC_H263_LEVEL_45 See IMP4HDVICPENC_H263Level enumeration for details. Default value is IMP4HDVICPENC_H263_LEVEL_40.
useVOS	XDAS_Int32	Input	MPEG-4 Visual sequence header ON/OFF ❑ 1 - ON ❑ 0 - OFF This feature is applicable for MPEG-4 mode only. Default value is 1.
useGOV	XDAS_Int32	Input	MPEG-4 GOV header ON/OFF ❑ 1 - ON ❑ 0 - OFF This feature is applicable for MPEG-4 mode only. Default value is 0.
useDataPartition	XDAS_Int32	Input	Data partition mode on/off. Data partition mode can be switched on only when packet size is greater than zero and within the maximum limit specified by MPEG-4 standard Annex-N ❑ 1 - ON ❑ 0 - OFF This feature is applicable for MPEG-4 mode only. This mode is supported up to CIF resolution and in high quality mode (i.e EncQuality_mode=0) only. For higher resolutions and in standard quality code (i.e EncQuality_mode=1), it is forced to OFF. Default value is 0.

Field	Datatype	Input/ Output	Description
useRVLC	XDAS_Int32	Input	RVLC mode ON/OFF. This mode can be switched on only when data partition mode is ON. ❑ 1 - ON ❑ 0 - OFF This mode is supported up to CIF resolution and in high quality mode only. For higher resolutions and in standard quality mode (i.e EncQuality_mode = 1), it is forced to OFF. This feature is applicable for MPEG-4 mode only. Default value is 0.
aspectRatio	XDAS_Int32	Input	Aspect ratio to be put in header See IMP4HDVICPENC_AspectRatio enumeration for details. Default value is IMP4HDVICPENC_AR_DEFAULT.
pixelRange	XDAS_Int32	Input	Pixel range to be put in header See IMP4HDVICPENC_PixelRange enumeration for details. Default value is IMP4HDVICPENC_PR_DEFAULT.
timerResolution	XDAS_Int32	Input	Timer resolution used for time stamp calculations. No of ticks per second. This should be greater than or equal to maximum frame rate in fps, and the value should be greater than 1 and less than 65535. Default value is 30000.
ME_Type	XDAS_Int32	Input	Motion Estimation algorithm type to be used by encoder. ❑ 0 - Normal search algorithm ❑ 1 - Low power search algorithm Default value is 0. Note: Low power ME search algorithm has reduced search points and may reduce the quality

Field	Datatype	Input/ Output	Description
UMV	XDAS_Int32	Input	Switch to enable/disable unrestricted MV option for encoding. ❑ 0 – UMV is disabled ❑ 1 – UMV is enabled Default value is 1. Note: By default, UMV is enabled inside the encoder if 4MV mode is enabled. (four_MV_mode is 1) irresepective of application setting. Disabling UMV may result in reduction of quality and improve performance marginally. It is recommended to disable UMV only for use cases where performance target is critical and not fulfilled.
EncQuality_mode	XDAS_Int32	Input	Switch to enable or disable Standard Quality mode for encoding. ❑ 0-High quality is enabled ❑ 1-standard quality is enabled Default value is 0. Note: Using standard quality mode may reduce the quality but performance is improved.
hdvicpHandle	Void	Input	Pointer to the HDVICP handle structure it is reserved and application can initialize to NULL

Note:

- Default values of extended parameters are used when size fields are set to the size of base structure IVIDENC1_Params.
- In case of MPEG-4 encoding, AC prediction is by default supported for resolutions up to CIF (352x288) and switched OFF for other resolutions. Also, AC prediction is swicthed OFF when levelIdc is IMP4HDVICPENC_SP_LEVEL_0 or IMP4HDVICPENC_SP_LEVEL_0B.

Force Intra MB updating is not supported by both H.263 and MPEG-4 modes as per Section 4.4 of H.263 standard and section A.1 in Annex-A of MPEG4 standard. Force updating ensures that each macro block is coded as INTRA atleast once every 132 times, when co-efficients are transmitted for that macroblock in P-pictures to control accumulation of inverse transform error.

4.2.2.2 IMP4HDVICPENC_DynamicParams

Description

This structure defines the run-time parameters and any other implementation specific parameters for a MPEG4 Encoder instance object. The run-time parameters are defined in the XDM data structure, `IVIDENC1_DynamicParams`.

Fields

Field	Datatype	Input/ Output	Description
`videncDynamicParams`	`IVIDENC1_DynamicParams`	Input	See `IVIDENC1_DynamicParams` data structure for details.
`Four_MV_mode`	`XDAS_Int32`	Input	Four MV mode on/off ❑ 1 - ON ❑ 0 - OFF This mode is supported up to VGA(640X480) resolution and in high quality mode `(i.e EncQuality_mode=0)` only. For higher resolutions and in standard quality code `(i.e EncQuality_mode=1)`, it is forced to OFF. This feature is applicable for MPEG-4 mode only and by default this is switched off at the time of encoder object creation. Default value is 0.
`PacketSize`	`XDAS_Int32`	Input	Insert resync marker (RM) based on number of bits specified by this parameter. A value of zero implies do not insert packets (RM). A non zero value implies distance between two successive resync markers in a coded frame will not exceed `packetSize` in bits. Minimum packet size is 1024 bits. There is no maximum limit when Data Partitioning is OFF. When Data Partitioning is ON, maximum limit specified by MPEG-4 standard Annex-N is applicable. Packet mode is supported up to CIF resolution only. For higher resolutions, it is forced to OFF. This feature is applicable for MPEG-4 mode only and by default, this is set to 0 at the time of encoder object creation. Default value is 0.
`qpIntra`	`XDAS_Int32`	Input	Default Quantization Parameter (QP) for I frame. This value must be between 2 and 31 for MPEG-4 and between 8 and 31 for H.263 By default, this is set to 8 at the time of encoder object creation.

Field	Datatype	Input/ Output	Description
qpInter	XDAS_Int32	Input	Quantization parameter for P frame. This value must be between 2 and 31 for MPEG-4 and between 8 and 31 for H.263. By default, this is set to 8 at the time of encoder object creation.
airRate	XDAS_Int32	Input	Adaptive intra refresh rate in no. of MBs per frame. Supported values to 0 to (No. of Mbs in frame-1) This feature is supported up to D1(720X576) resolution only. For higher resolutions it is forced to 0 and by default, this is set to 0 at the time of encoder object creation.
useHEC	XDAS_Int32	Input	HEC mode on./off ❑ 1 - ON ❑ 0 - OFF This feature is supported upto CIF resolution only. For higher resolutions, it is forced to OFF. This feature is applicable for MPEG-4 mode only and by default, this is set to OFF at the time of encoder object creation. HEC mode can be switched ON only when packet mode is ON.
useGOBSync	XDAS_Int32	Input	Indicates whether or not to insert GOB headers in the bit stream as defined by the standard values : ❑ 0 – No GOB headers ❑ 1- Insert GOB headers This feature is applicable for H263 mode only and by default, this is set to 0 at the time of encoder object creation.
RcAlgo	XDAS_Int32	Input	Rate control algorithm to be used. See IMP4HDVICPENC_RCAlgo enumeration for details. Rate Control algorithm can be changed only before first encode call (that is, process()call). Any call to change rate control algorithm during the encoding process will be ignored by encoder. Default value is IMP4HDVICPENC_RC_DEFAULT. Note: All the supported RC algorithms do not support quantization scale variation within the I-frames at row level .

Field	Datatype	Input/ Output	Description
QPMax	XDAS_Int32	Input	Max Quantization Parameter (QP) to be used by encoder. This value must be between 2 and 31 for MPEG-4 and between 8 and 31 for H.263. This should be greater than QPMin. By default, this is set to 31 at the time of encoder object creation.
QPMin	XDAS_Int32	Input	Max Quantization Parameter (QP) to be used by encoder. This value must be between 2 and 31 for MPEG-4 and between 8 and 31 for H.263. This should be less than QPMax. By default, this is set to 2 at the time of encoder object creation. For H.263, this will be changed to 8 at run-time.
maxDelay	XDAS_Int32	Input	Maximum acceptable delay in milliseconds for rate control. This value should be greater than 100 ms. Currently, there is no maximum limit for this parameter. However, the application can use up to 10000 ms. Typical value is 1000 ms. By default, this is set to 1000 ms at the time of encoder object creation.
qpInit	XDAS_Int32	Input	Initial Quantization Parameter for first frame (QP) to be used by encoder when rcAlgo is not equal to IMP4HDVICPENC_RC_NONE and bit-rate is a non-zero value. This value must be between 2 and 31 for MPEG-4 non-packet mode, between 4 and 31 for MPEG-4 packet mode and between 8 and 31 for H.263. The encoder will clip any other value to the limits mentioned above. qpInit value can be changed only before the first encode call (that is, process() call). Any call to change qpInit value during the encoding process will be ignored by encoder. For low delay / low bit-rate CBR applications it is recommended to use a high value of qpInit (> 20) so as to avoid decoder VBV underflow errors. This is because the first frame uses a constant quantizer specified by qpInit which may generate encoded stream that exceeds the maximum VBV buffer size. By default, this is set to 8 at the time of encoder object creation.
PerceptualRC	XDAS_Int32	Input	This is a reserved field and only the value of zero is supported.

Field	Datatype	Input/ Output	Description
reset_vIMCOP_every_frame	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 is 1. If this flag is set, MPEG4 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 MPEG4 decoder or if running another MPEG4 encoder instance with different quality settings encodingPreset and Encquality_mode. This is because MPEG4 encoder uses different HDVICP code and data for standard quality and high quality. However, application can set this flag to 0 for better performance if it runs multiple instances of MPEG4 encoder with same quality settings.
mvSADoutFlag	XDAS_Int32	Input	Flag to enable/disable MVSAD info to the application. ❑ 1- enable exposing MV SAD info to application ❑ 0- Disable exposing MV SAD info to application

Note:

- Default values of extended parameters are used when size fields are set to the size of base structure IVIDENC1_DynamicParams.
- Rate control algorithms supported by this version of encoder may deviate fromVBV buffer size as mentioned in MPEG-4 standard Annex-N for corresponding level because size is very small for higher resolutions like 720p, 1080p and so the VBV buffer size is calculated as vbv_buffer_size = (maxDelay * pMP4Enc->bit_rate) /1000 Irrespective of the limitation imposed by the standard on the given level.

4.2.2.3 IMP4HDVICPENC_exportMEdata

‖ Description

This structure defines structure for exporting ME info: Motion vectors and ME Sad.

‖ Fields

Field	Data type	Input/ Output	Description
meSad	XDAS_Int32	Output	ME Sad value of the MB to be stored
mvX	XDAS_Int16	Output	Motion vector along x for the MB under consideration. Stored in full pel resolution
mvY	XDAS_Int16	Output	Motion vector along y for the MB under consideration. Stored in full pel resolution

4.2.2.4 IMP4HDVICPENC_InArgs

Description

This structure defines the run-time input arguments for MPEG4 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 put in bit-stream. This should be integral multiple of Timer Resolution/ (Frame rate in fps). Initial time stamp value (for first frame) should be 0. Default value is calculated as Frame number * Timer Resolution/ (Frame rate in fps). For H.263, timer resolution is assumed as 30. The example for calculation of time stamp value is shown in the test application. However, the application can provide timer value from capture device to the encoder. The encoder will use this value in the bit-stream without any changes.

Note:

Default values of extended parameters are used when size fields are set to size of base structure IVIDENC1_InArgs.

4.2.2.5 IMP4HDVICPENC_Status

Description

This structure defines parameters that describe the status of the MPEG4 Encoder and any other implementation specific parameters. The status parameters are defined in the XDM data structure, `IVIDENC1_Status`.

Fields

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

4.2.2.6 IMP4HDVICPENC_OutArgs

Description

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

Fields

Field	Data type	Input/ Output	Description
`videncOutArgs`	`IVIDENC1_OutArgs`	Output	See `IVIDENC1_OutArgs` data structure for details.
`packetSize`	`XDAS_Int32*`	Output	Buffer to output individual packet sizes in bytes. Application should allocate the buffer with size of (396 * 4) bytes and send the pointer to encoder. 396 is the maximum number of packets possible in one CIF frame, which is equal to the number of MBs in the frame. This is used as the maximum number of packets as this encoder version supports packet mode only up to CIF resolution. Only sizes of valid packets indicated by `numPackets` will be filled by encoder and the remaining values in the buffer are invalid. Note: The first packet size of each frame includes VOS, VOL, GOV, and VOP header sizes, if they are present in the bit-stream.
`numPackets`	`XDAS_Int32`	Output	Total number of packets in the encoded frame.

4.2.2.7 IMP4HDVICPENC_Fxns

Description

This structure defines all of the operations for the MPEG4 Encoder instance object.

Fields

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

4.3 Interface Functions

This section describes the Application Programming Interfaces (APIs) used in the MPEG4 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).

4.3.1 Creation APIs

Creation APIs are used to create an instance of the component. The term creation could mean allocating system resources, typically memory.

‖ Name

`algNumAlloc()` – determine the number of buffers that an algorithm requires

‖ Synopsis

```
XDAS_Int32 algNumAlloc(Void);
```

‖ Arguments

```
Void
```

‖ Return Value

```
XDAS_Int32; /* number of buffers required */
```

‖ Description

`algNumAlloc()` returns the number of buffers that the `algAlloc()` method requires. This operation allows you to allocate sufficient space to call the `algAlloc()` method.

`algNumAlloc()` may be called at any time and can be called repeatedly without any side effects. It always returns the same result. The `algNumAlloc()` API is optional.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (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()`

4.3.2 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()`

4.3.3 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:

If extended data structures are used, 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. To use base parameter structure set the `size` field o size of base data structure and send the third and fourth arguments accordingly. Depending on the value set for the `size` field, the algorithm uses either basic or extended parameters. If application sends base structure, algorithm will set default values for all 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, mpeg4venc_ti_arm926testapp.c available in the \client\test\src sub-directory.

‖ See Also

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

4.3.4 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).

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, mpeg4venc_ti_arm926testapp.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 MPEG4 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()`

4.3.5 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.

Appendix A

Revision History

This user guide revision history highlights the changes made to SPRUGR4 codec specific user guide to make it SPRUGR4A.

Table A-1. Revision History for MPEG4 HDVICP Simple Profile Encoder on DM365

Section	Changes
Section 2.4.2	Encoder Configuration File ❑ Modified sample testparams.cfg file
Section 4.1.2	MPEG4 Encoder Symbolic Constants and Enumerated Data Types Removed following Group or Enumeration Class ❑ IMP4VENC_STATUS ❑ IMP4VENC_H263Level ❑ IMP4VENC_CmdId ❑ IMP4VENC_Level ❑ IMP4VENC_PixelRange ❑ IMP4VENC_ChromaFormat ❑ IMP4VENC_ChromaFormat ❑ IMP4VENC_RCAlgo Added following Group or Enumeration Class ❑ IMP4HDVICPENC_RCAlgo ❑ IMP4HDVICPENC_ChromaFormat ❑ IMP4HDVICPENC_AspectRatio ❑ IMP4HDVICPENC_PixelRange ❑ IMP4HDVICPENC_CmdId ❑ IMP4HDVICPENC_Level ❑ IMP4HDVICPENC_H263Level ❑ IMP4HDVICPENC_STATUS

Section	Changes
Section 4.2.2	MPEG4 Encoder Data Structures Removed following Data Structures ❑ IMP4VENC_Params ❑ IMP4VENC_DynamicParams ❑ IMP4VENC_exportMEdata ❑ IMP4VENC_InArgs ❑ IMP4VENC_Status ❑ IMP4VENC_OutArgs ❑ IMP4VENC_Fxns Added following Data Structures ❑ IMP4HDVICPENC_Params ❑ IMP4HDVICPENC_DynamicParams ❑ IMP4HDVICPENC_exportMEdata ❑ IMP4HDVICPENC_InArgs ❑ IMP4HDVICPENC_Status ❑ IMP4HDVICPENC_OutArgs ❑ IMP4HDVICPENC_Fxns