MPEG2 Main Profile Encoder on DM365

User's Guide

Literature Number: SPRUGS9A
September 2010

Preface

Read This First

About This Manual

This document describes how to install and work with Texas Instruments' (TI) MPEG2 Main 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 – Time-Stamp Insertion**, describes insertion of frame time-stamp.

- **Appendix B - Revision History**, highlights the changes made to SPRUGS9 codec specific user guide to make it SPRUGS9A

Related Documentation from Texas Instruments

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

- *TMS320 DSP Algorithm Standard Rules and Guidelines* (SPRU352) defines a set of requirements for DSP algorithms that, if followed, allow system integrators to quickly assemble production-quality systems from one or more such algorithms.
- *TMS320 DSP Algorithm Standard API Reference* (SPRU360) describes all the APIs that are defined by the TMS320 DSP Algorithm Interoperability Standard (also known as XDAIS) specification.
- *Using IRES and RMAN Framework Components for C64x+* (literature number SPRAAI5) provides an overview of the IRES interface, along with some concrete resource types and resource managers that illustrate the definition, management and use of new types of resources.

Related Documentation

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

- *ISO/IEC 13818-2:2000 Second Edition, Information Technology –* Generic coding of moving pictures and associated audio information: Video

MPEG2 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
CCS	Code Composer Studio
D1	720x480 or 720x576 Resolutions in Progressive Scan
DC	Direct Current
DCT	Discrete Cosine Transform

Abbreviation	**Description**
DP	Data Partitioning
DM	Digital Media
DMA	Direct Memory Access
DSP	Digital Signal Processor
HDVICP	High Definition Video and Imaging Co-Processor sub-system
IDCT	Inverse Discrete Cosine Transform
MB	Macro Block
ME	Motion Estimation
MPEG	Motion Pictures Expert Group
MV	Motion Vector
NTSC	National Television Standards Committee
PDM	Parallel Debug Manager
RTOS	Real Time Operating System
VGA	Video Graphics Array
VLC	Variable Length Coding
XDAIS	eXpressDSP Algorithm Interface Standard
XDM	eXpressDSP Digital Media
YUV	Color space in luminance and chrominance form

Text Conventions

The following conventions are used in this document:

❑ Text inside back-quotes (‘‘) represents pseudo-code.

❑ Program source code, function and macro names, parameters, and command line commands are shown in a `mono-spaced` font.

Product Support

When contacting TI for support on this codec, quote the product name (MPEG2 Main 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 MPEG2 Main Profile Encoder on the DM365 platform and its supported features.

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. Framework Component acts as a software interface between the perating system and the codec. Framework Component manages resources and memory by interacting with kernel through predefined APIs.

Following diagram shows the software architecture.

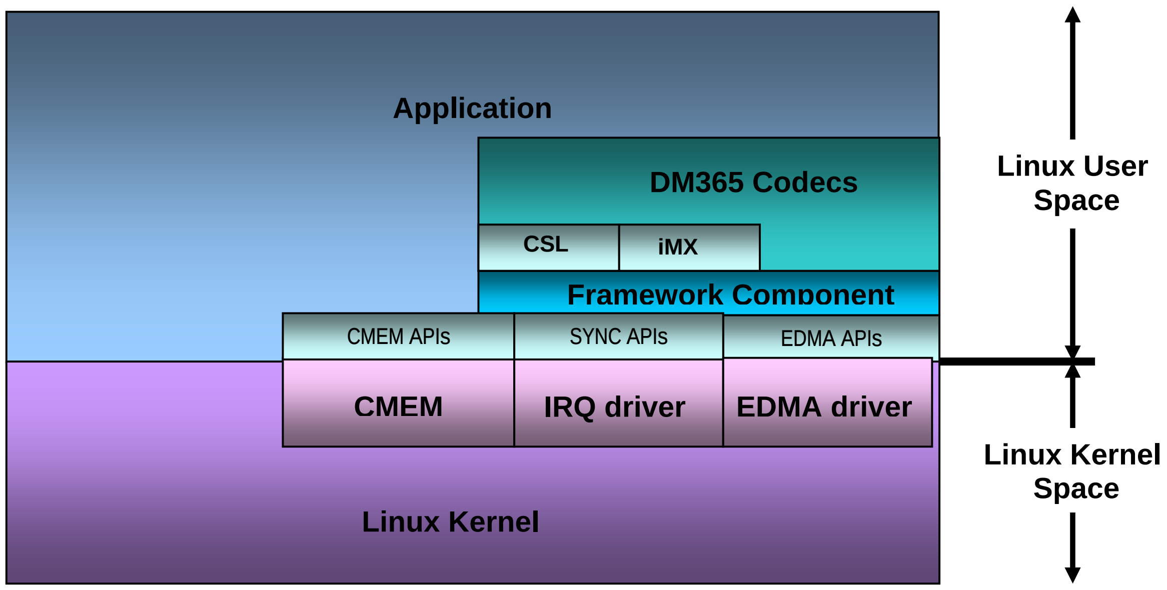


Figure 1-1. Software Architecture.

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

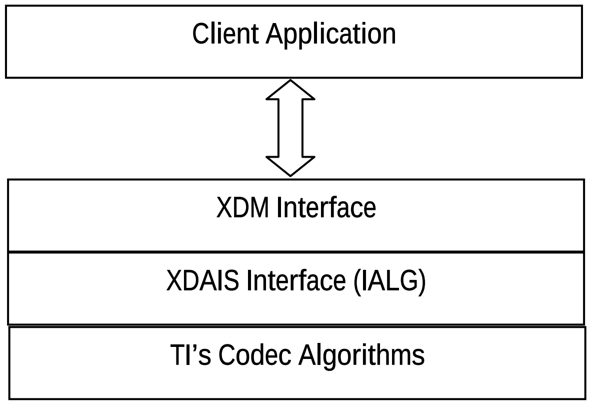
In the multimedia application space, you have the choice of integrating any codec into your multimedia system. For example, if you are building a video decoder system, you can use any of the available video decoders (such as MPEG4, H.263, or MPEG2) in your system. To enable easy integration with the 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, It also serves as a resource manager. The following block diagram shows the FC components and their interfacing structure.

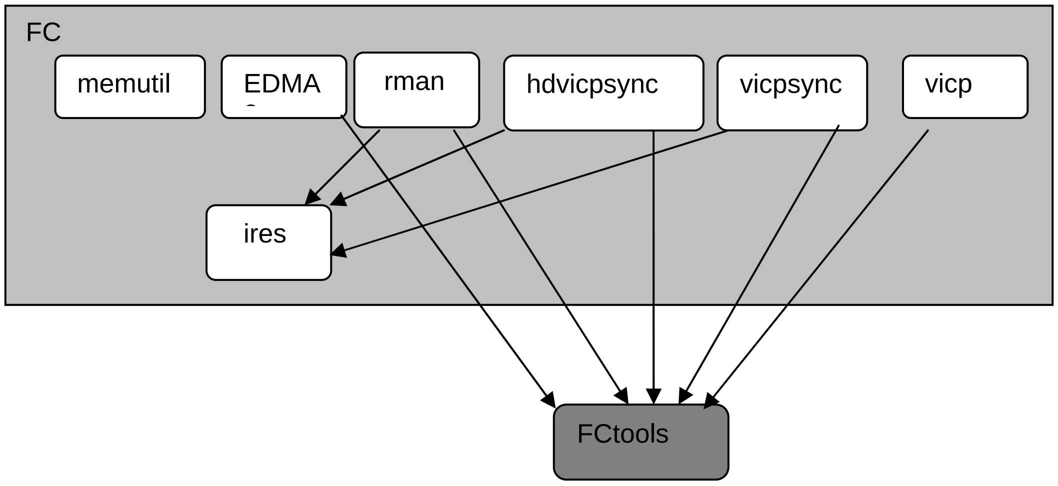


Figure 1-2. Framework Component Interfacing Structure.

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).

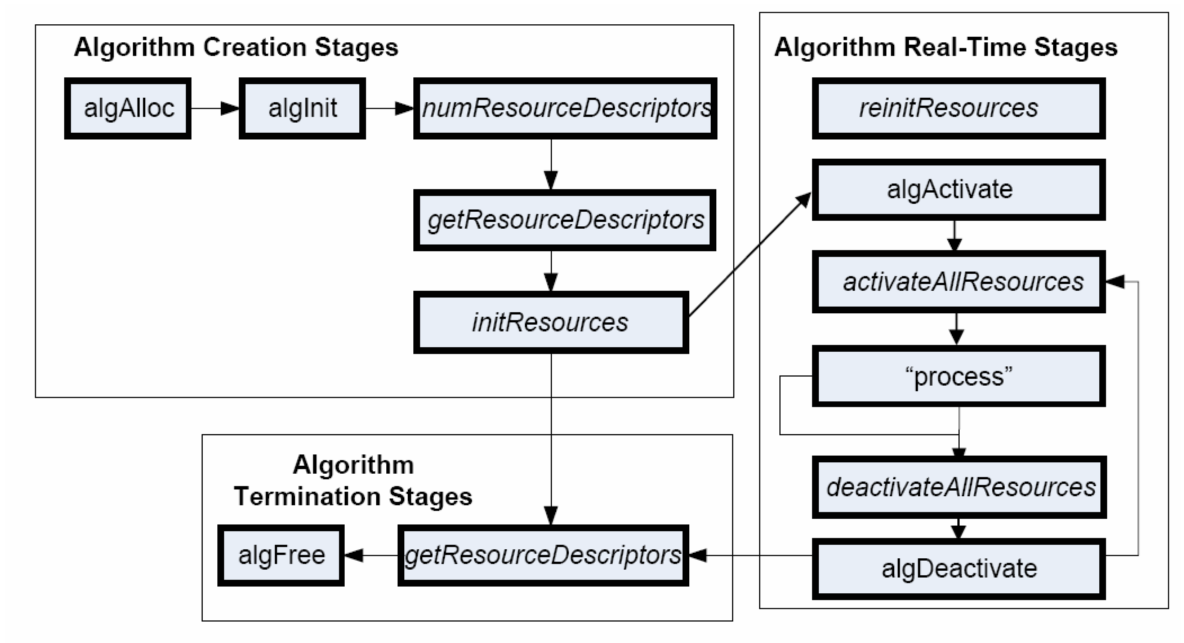


Figure 1-3. IRES Interface Definition and Function-calling Sequence.

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 MPEG2 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 MPEG2 Main Profile Encoder

MPEG2(from ISO/IEC) is a popular video coding algorithm enabling high quality multimedia services on a limited bandwidth network. MPEG2 standard defines several profiles and levels that specify restrictions on the bit-stream and hence, limits the capabilities needed to encode/decode the bit-streams. Each profile specifies a sub-set of algorithmic features and limits encoders conforming to that profile. Each level specifies a set of limits on the values that can be taken by the syntax elements in the profile.

Some important features of MPEG2 Main Profile @ High Level are:

- ❑ Main Profile:
  - Only I and P type Pictures are present
  - Both Interlace and Progressive pictures types are present
  - Motion Estimation and Compensation pixel accuracy up to half-pixel

Figure 1-4 depicts the working of the encoder.

The input video sequence for MPEG2 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 MPEG2 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.

The following figure depicts the working of the encoder.

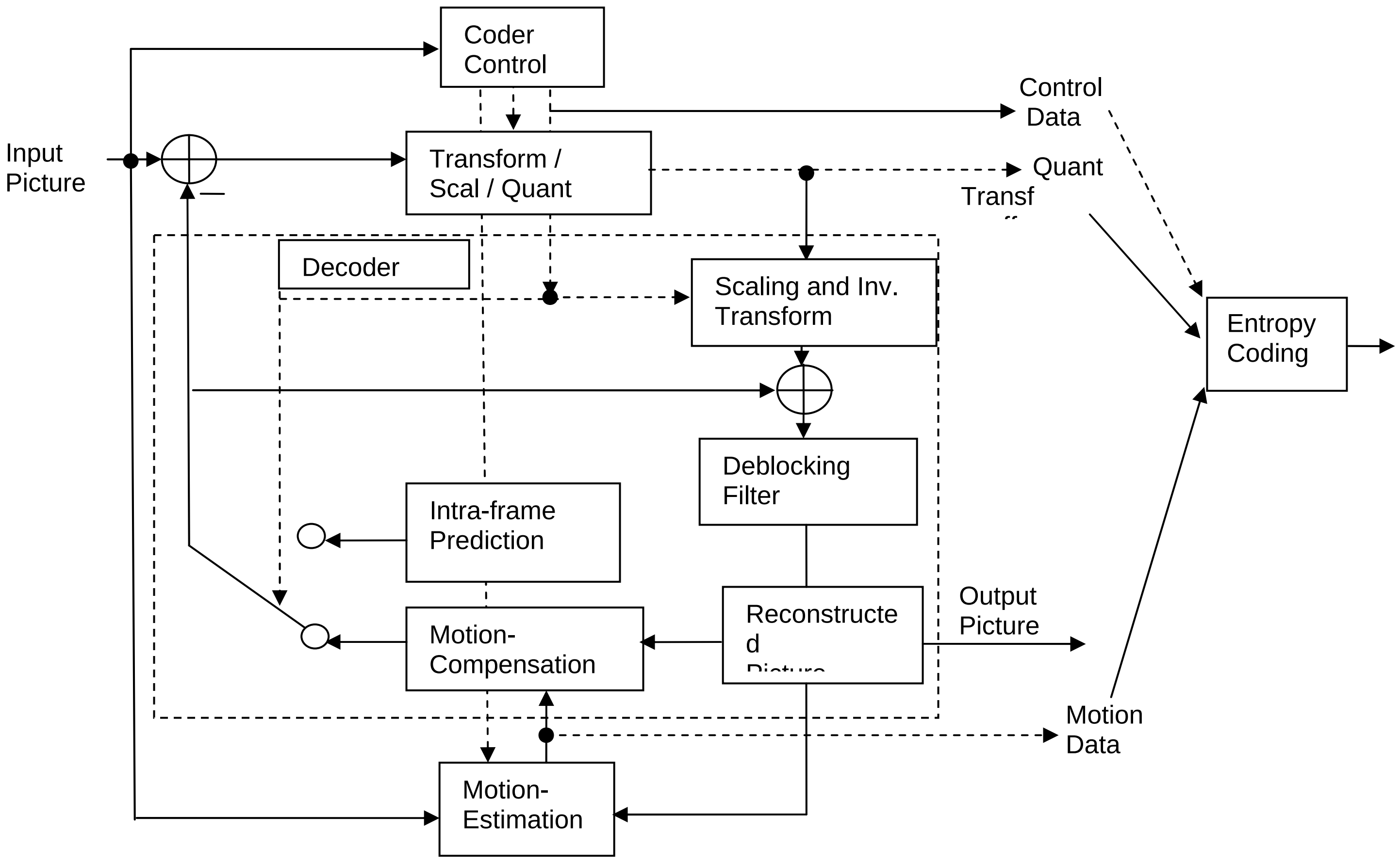


Figure 1-4. Block Diagram of MPEG2 Encoder.

From this point onwards, all references to MPEG2 Encoder mean MPEG2 Main Profile (MP) Encoder only.

1.4 Supported Services and Features

This user guide accompanies TI's implementation of MPEG2 Encoder on the DM365 platform.

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

- eXpressDSP Digital Media (XDM1.0 IVIDENC1) interface compliant
- Compliant with MPEG2 Main Profile up to high level
- Supports resolutions up to 1920x1088
- Supports YUV420 semi planar input format for the frames
- Supports progressive and interlaced (field pictures only) encoding
- Generates bit-stream compliant with MPEG2 standard
- Supports only 16x16 MB partition
- Supports frame based encoding with frame size being multiples of 2

- Supports rate control (CBR and VBR)
- Supports Half Pel Interpolation for motion estimation

This version of the codec does not support following features of the standard:

- Frame picture encoding in Interlace
- Adaptive frame/field macro block decisions
- Field DCT

DM365 MPEG2 encoder can be configured in two modes:

- Standard quality, standard feature, which gives performance of 720P@39fps
- High quality, full feature, which is for 720p@25fps.

Supported features in High quality mode:

- Supports TI's proprietary motion estimation algorithm (Full search ME)

Supported features in standard quality mode:

- Supports TI's proprietary motion estimation algorithm (Low power ME)

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_mpeg2enc_xx_xx_xx_xx_production. Figure 2-5 shows the sub-directories created in this directory.

Note:

xx_xx_xx_xx in the directory name is the version of the codec. For example, If the version of the codec is 02.00.01.00, then the directory created on extraction of tar file is dm365_mpeg2enc_02_00_01_00_production.

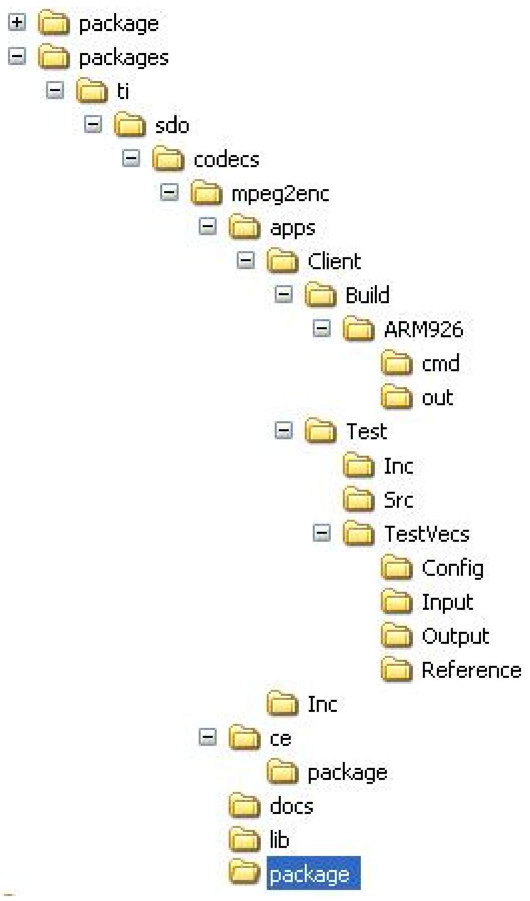


Figure 2-5. Component Directory Structure for Linux.

Table 2-2 provides a description of the sub-directories created in the dm365_mpeg2enc_xx_xx_xx_xx production directory.

Table 2-2. Component Directories for Linux.

Sub-Directory	Description
\package	Contains files related while building the package
\packages\ti\sdo\codecs\mpeg2enc\lib	Contains the codec library files on host
\packages\ti\sdo\codecs\mpeg2enc\docs	Contains user guide, datasheet, and release notes
\packages\ti\sdo\codecs\mpeg2enc\apps\client\build\arm926	Contains the make file to built sample test application
\packages\ti\sdo\codecs\mpeg2enc\apps\client\build\arm926\cmd	Contains a template (.xdt) file to used to generate linker command file
\packages\ti\sdo\codecs\mpeg2enc\apps\client\build\arm926\map	Contains the memory map generated on compilation of the code

Sub-Directory	Description
\packages\ti\sdo\codecs\mpeg2enc\apps\client\test\src	Contains application C files
\packages\ti\sdo\codecs\mpeg2enc\apps\client\test\inc	Contains header files needed for the application code
\packages\ti\sdo\codecs\mpeg2enc\apps\client\test\testvecs\input	Contains input test vectors
\packages\ti\sdo\codecs\mpeg2enc\apps\client\test\testvecs\output	Contains output generated by the codec
\packages\ti\sdo\codecs\mpeg2enc\apps\client\test\testvecs\reference	Contains read-only reference output to be used for verifying against codec output
\packages\ti\sdo\codecs\mpeg2enc\apps\client\test\testvecs\config	Contains configuration parameter files

2.3 Building and Running the Sample Test Application on Linux (HDVICP)

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\mpeg2enc\lib.

2) Verify that codec object library library mpeg2venc_ti_arm926.a exists in the \packages\ti\sdo\codecs\mpeg2enc\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\mpeg2enc\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\mpeg2enc\client\build\arm926 and type the command make. This generates an executable file mpeg2venc-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 ./mpeg2venc-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\mpeg2enc\apps\client\test\testvecs\config sub-directory.

The format of the testvecs_linux.cfg file is:

```
X
config
input
output/reference
recon

export_me_info
```

where:

- ❑ `X` may be set as:
  - o 1 - for compliance checking, no output file is created
  - o 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).
- ❑ `export_me_info` is output ME information file

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_5frms_420p.m2v
../../test/testvecs/output/colorful_toys_cif_5frms_420p_recon.yuv
../../test/testvecs/output/colorful_toys_cif_5frms_420p_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
#########################################################################
ImageWidth = 352         # Image width in Pels
ImageHeight = 288        # Image height in Pels
captureWidth = 0         # Image Capture Width
FrameRate = 30000        # Frames per second (fps)*1000
timerResolution = 30     # Timer Resolution for Picture Timing
mvSADout = 1              # Flag to enable/disable exporting of ME output
BitRate = 512000         # Bitrate(bps)
ChromaFormat = 9         # 256 => IMP2VENC_YUV_420IUV(only supported value)
RCAlgo = 8               # 0 = IMP2VENC_RC_NONE ,
                         # 4 - IMP2VENC_RC_CBR,
                         # 8- IMP2VENC_RC_VBR
maxDelay = 1000          # Delay parameter fir Ratecontrol 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
IntraPeriod = 40         # Period of I-Frames
Intra_QP = 2
Inter_QP = 2
QPMax = 31         # Max Qp value - range {1, 31}
QPMin = 1          # Max Qp value - range {1, 31}
InitQp   = 2
QscaleType = 0             # 0 => Range of Qp = {1,62}
                           #1 => Range of Qp = {1,112}
IntraDCPrec = 0            # 0 => 8 bits; 1 => 9 bits; 2 => 10 bits;
                           # 3 => 11 bits
Interlace = 0               # IVIDEO_PROGRESSIVE = 0 or
                            # IVIDEO_INTERLACED = 1
FramesToEncode = 5         # Number of frames to be encoded
ME_Type = 0                 # 1 = Normal Search, 1 = Low Power
PerceptualRC = 0            #  Only 0 is supported. Currently this is a
                                                 reserved field
EncoderPreset = 3           # 0 = XDM_DEFAULT is currently supported
RateControlPreset = 5      # IVIDEO_LOW_DELAY = 1, /**< CBR rate control
                                               for video conferencing. */
                           # IVIDEO_STORAGE = 2, /**< VBR rate control for
                                                    local storage (DVD)
                                                    recording.     */
                           # IVIDEO_TWOPASS = 3,    /**< Two pass rate
                                                     control for non real
                                                     time applications.   */
                           # IVIDEO_NONE = 4,  /**< No configurable
                                                video rate control
                                                mechanism.*/
                           # IVIDEO_USER_DEFINED = 5,/**< User defined
                                                    configuration using
                                                     extended parameters.*/
                           # IVIDEO_RATECONTROLPRESET_DEFAULT =
                              IVIDEO_LOW_DELAY /**< Default setting.*/
```

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 2.3.

2.4.3 Encoder Sample Base Param Setting

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

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

2.5 Standards Conformance and User-Defined Inputs

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

To check the conformance of the codec for other input files of your choice, follow these steps:

1) Copy the input files to the \client\test\testvecs\input sub-directory.
2) Copy the reference files to the \client\test\testvecs\reference sub-directory.
3) Edit the configuration file, testvecs_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.

3.1 Overview of the Test Application

The test application exercises the `IVIDENC1` base class of the MPEG2 Encoder library. The main test application files are mpeg2encoderapp.c and mpeg2encoderapp.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.

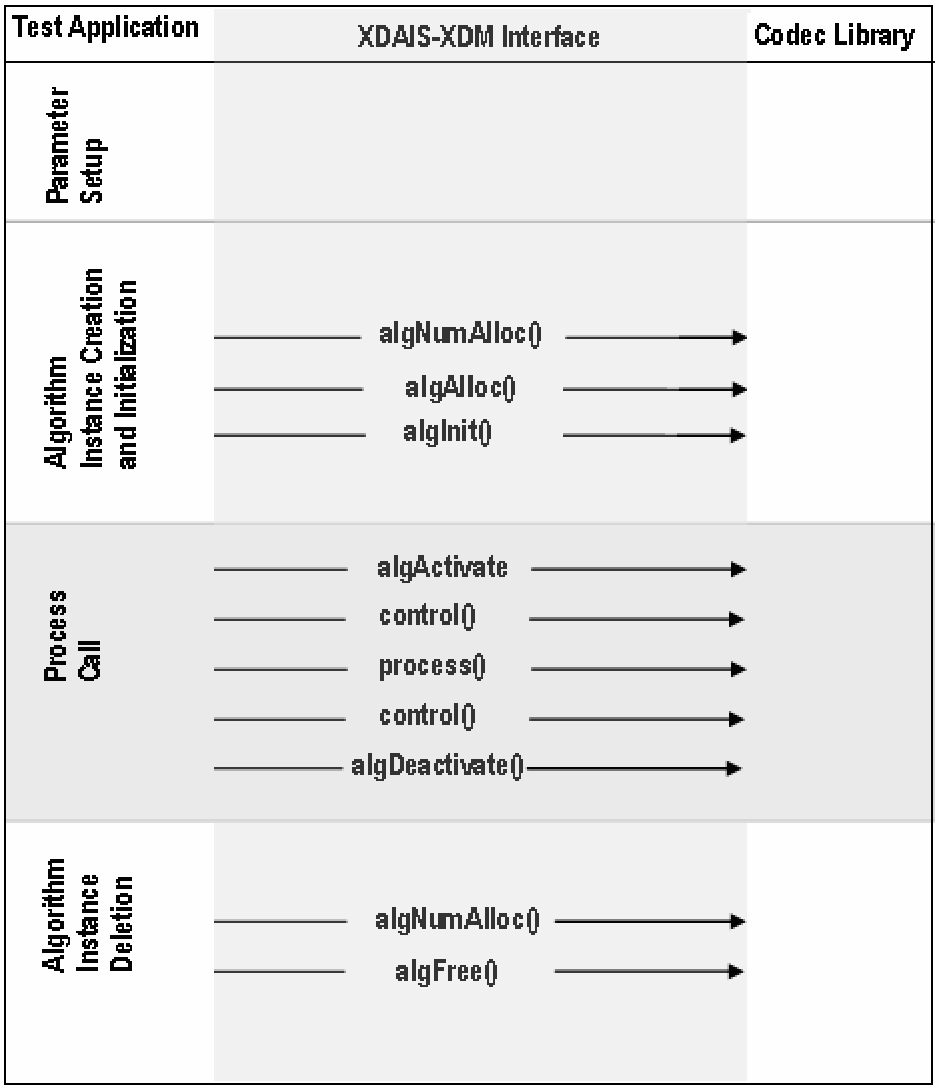


Figure 3-1. Test Application Sample Implementation

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 `IMPEG2VENC_Params` structure based on the values it reads from the Testparams.cfg file.
5) 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.
4) 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`.

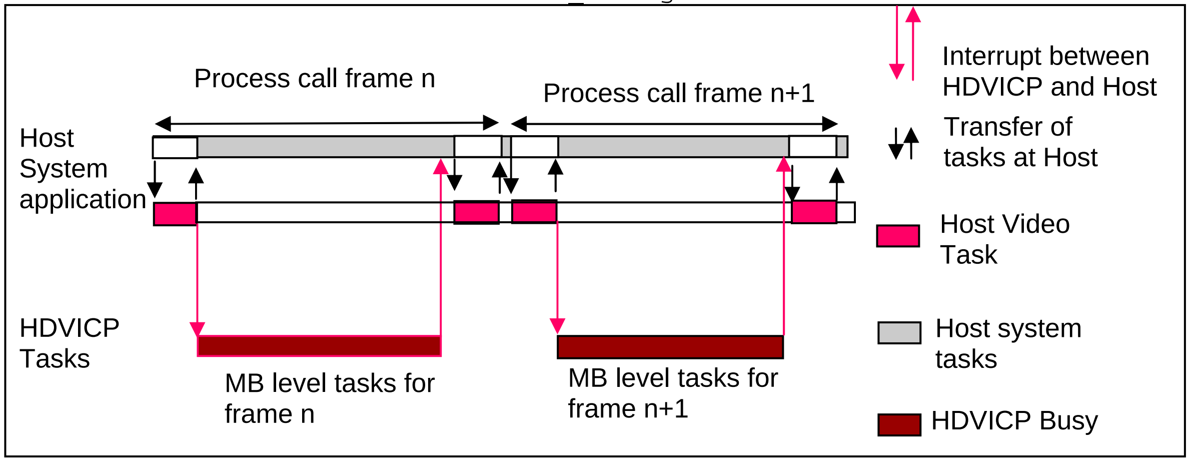


Figure 3-2. Process Call with Host Release

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

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

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

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

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

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

3.1.4 Algorithm Instance Deletion

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

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

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

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

3.2 Handshaking Between Application and Algorithm

3.2.1 Resource Level Interaction

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

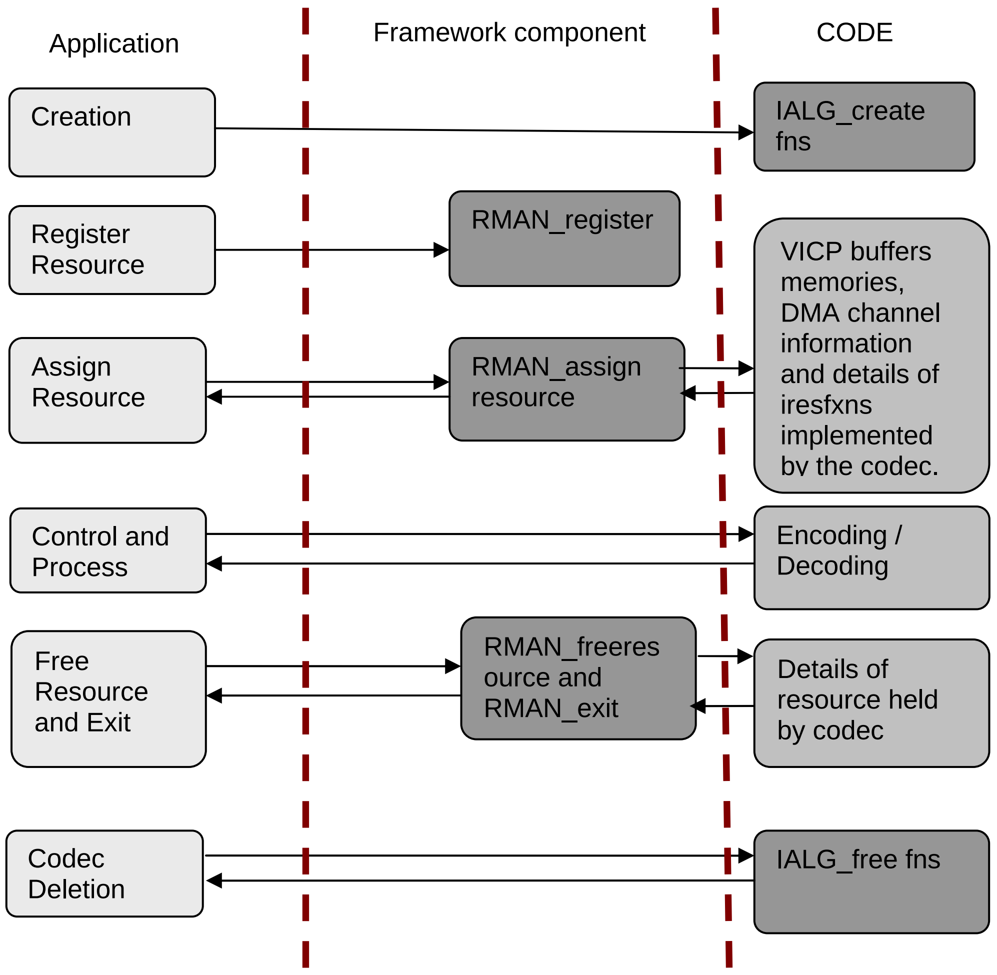
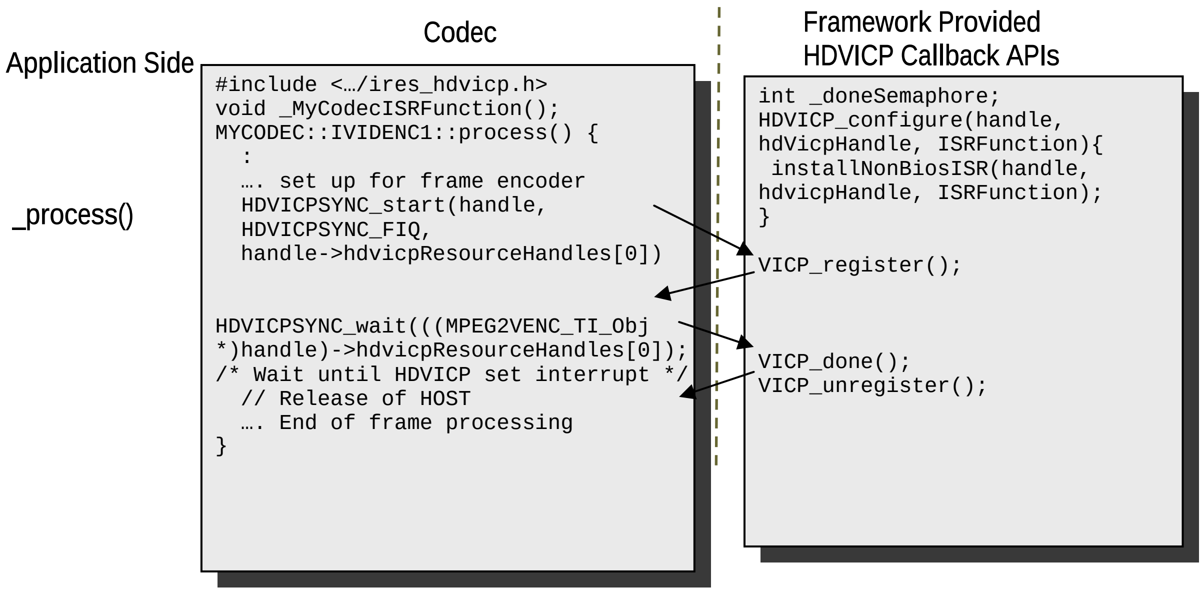


Figure 3-3. Resource Level Interaction.

3.2.2 Handshaking Between Application and Algorithms

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



```
#include <…/ires_hdvicp.h>
void _MyCodecISRFunction();
MYCODEC::IVIDENC1::process() {
  :
  …. set up for frame encoder
  HDVICPSYNC_start(handle,
  HDVICPSYNC_FIQ,
  handle->hdvicpResourceHandles[0])


HDVICPSYNC_wait(((MPEG2VENC_TI_Obj
*)handle)->hdvicpResourceHandles[0]);
/* Wait until HDVICP set interrupt */
  // Release of HOST
  …. End of frame processing
}
```

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

VICP_register();



VICP_done();
VICP_unregister();
```

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.

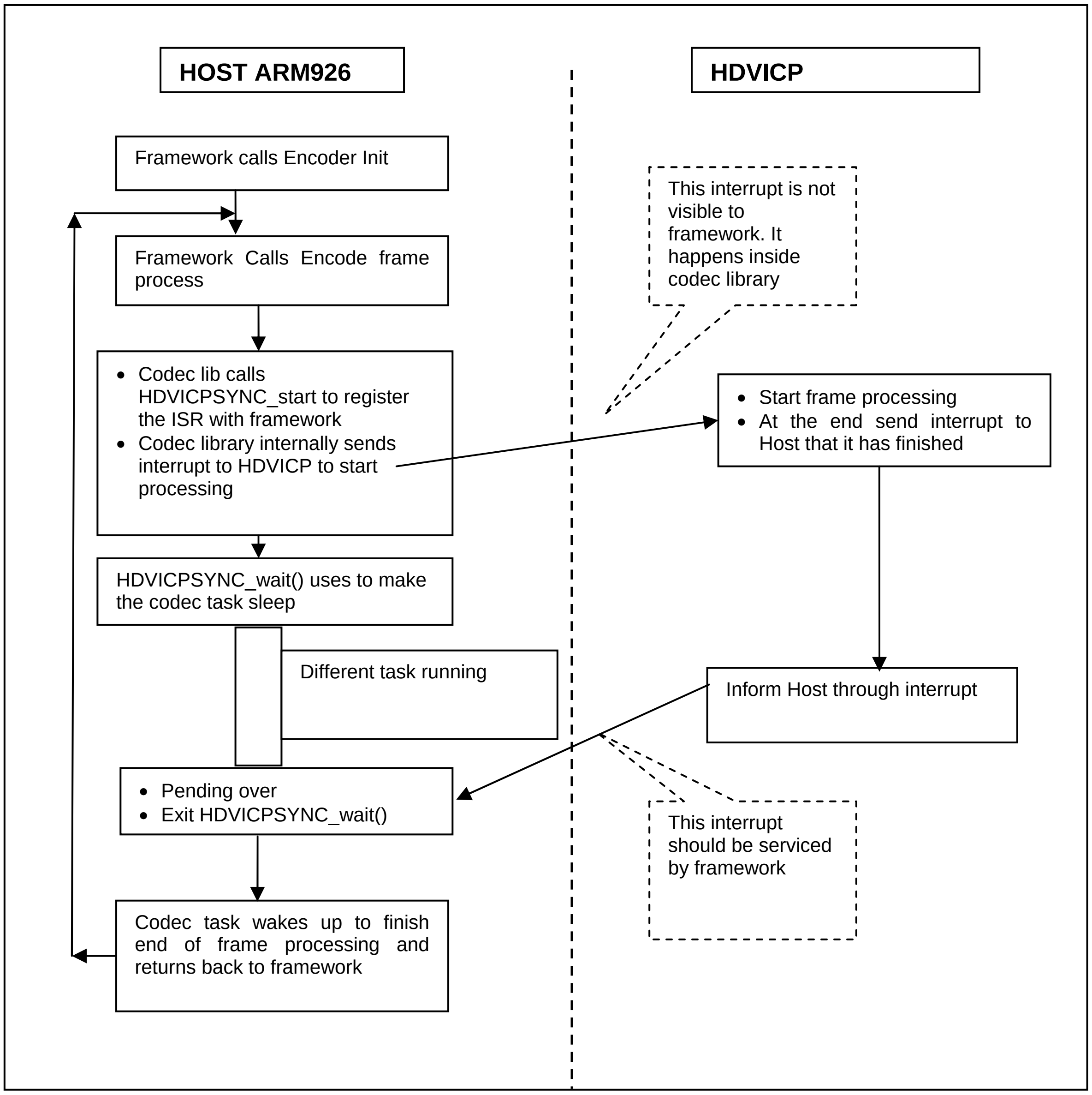

HOST ARM926
HDVICP
Framework calls Encoder Init
Framework Calls Encode frame process
This interrupt is not visible to framework. It happens inside codec library
• Codec lib calls HDVICPSYNC_start to register the ISR with framework
• Codec library internally sends interrupt to HDVICP to start processing
• Start frame processing
• At the end send interrupt to Host that it has finished
HDVICPSYNC_wait() uses to make the codec task sleep
Different task running
Inform Host through interrupt
• Pending over
• Exit HDVICPSYNC_wait()
This interrupt should be serviced by framework
Codec task wakes up to finish end of frame processing and returns back to framework

Figure 3-5. Interrupt Between Codec and Application.

3.3 Cache Management by Application

3.3.1 Cache Usage By Codec Algorithm

The codec source code and data, which runs on Host ARM926 can be placed in DDR. The host of DM365 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 MPEG2 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 = MP2VENC_create(&fxns, &params)

/*Getting instance of algorithms that implements IALG and
IRES functions*/
iErrorFlag = RMAN_assignResources((IALG_Handle)handle,
                             &MPEG2VENC_TI_IRES, /* IRES_Fxns*
*/
                               1 /* scratchId */);

/* Get Buffer information  */
iErrorFlag = MP2VENC_control(
            handle,         // Instance Handle
            XDM_GETBUFINFO,  // Command
            &dynamicparams, // Pointer to Dynamicparam structure
            &status         // Pointer to the status structure
        );

/* Allocate memory for input and output frame buffers */
AllocateMP2IOBuffers(
    status, // status structure
    &inobj, // Pointer to Input Buffer Descriptor
    &outobj) // Pointer to Output Buffer Descriptor);

/*SET BASIC INPUT PARAMETERS */
iErrorFlag = MP2VENC_control(
            handle,         // Instance Handle
            XDM_SETPARAMS,  // Command
            &dynamicparams, // Pointer to Dynamicparam structure
            &status         // Pointer to the status structure
        );

/*Set Dynamic input parameters */
iErrorFlag = MP2VENC_control(
            handle,         // Instance Handle
            XDM_GETSTATUS,  // Command
            &dynamicparams, // Pointer to Dynamicparam structure
            &status         // Pointer to the status structure
        );
```

```
/* for Loop for encode Call  for a given no of frames */
for(;;)
{
    /* Read the input frame in the Application Input Buffer */
    ReadInputData (inFile);
    /*----------------------------------------------------*/
    /* Start the process : To start Encoding a frame      */
    /* This will always follow a MPEG2VENC_encode_end call */
    /*----------------------------------------------------*/

    iErrorFlag = MP2VENC_encode (
                handle,   // Instance Handle    - Input
                &inobj,   // Input Buffers      - Input
                &outobj,  // Output Buffers     - Output
                &inargs,  // Input Parameters   - Input
                &outargs  // Output Parameters  - Output
            );

/* Get the statatus of the Encoder using control */
MP2VENC_control(
                    handle,         // Instance Handle
                    XDM_GETSTATUS,  // Command - GET STATUS
                    &dynamicparams, // Input
                    &status         // Output
                );
} /* end of Do-While loop - which Encodes frames  */

/* Free Input and output buffers */
FreeMP2IOBuffers(
            &inobj, // Pointer to Input Buffer Descriptor
&outobj // Pointer to Output Buffer Descriptor  );

/* Free assigned resources */
RMAN_freeResources((IALG_Handle)(handle),
                    &MPEG2VENC_TI_IRES, /* IRES_Fxns* */
                   );
/* Delete the encoder Object handle*/
MP2VENC_delete(handle);

/* Unregister protocal*/
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.

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 MPEG2 Encoder.
	`IVIDEO_IDR_FRAME`	Intra coded frame that can be used for refreshing video content. Not supported in MPEG2 Encoder
	`IVIDEO_II_FRAME`	Interlaced frame, both fields are I frames..
	`IVIDEO_IP_FRAME`	Interlaced frame, first field is an I frame, second field is a P frame.
	`IVIDEO_IB_FRAME`	Interlaced frame, first field is an I frame, second field is a B frame. Not supported in this version of MPEG2 Encoder.
	`IVIDEO_PI_FRAME`	Interlaced frame, first field is a P frame, second field is an I frame. Not supported in this version of MPEG2 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 MPEG2 Encoder.
	`IVIDEO_BI_FRAME`	Interlaced frame, first field is a B frame, second field is an I frame. Not supported in this version of MPEG2 Encoder.
	`IVIDEO_BP_FRAME`	Interlaced frame, first field is a B frame, second field is a P frame. Not supported in this version of MPEG2 Encoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_BB_FRAME	Interlaced frame, both fields are B frames. Not supported in this version of MPEG2 Encoder.
	IVIDEO_MBAFF_I_FRAME	Intra coded MBAFF frame. Not supported in this version of MPEG2 Encoder.
	IVIDEO_MBAFF_P_FRAME	Forward inter coded MBAFF frame. Not supported in this version of MPEG2 Encoder.
	IVIDEO_MBAFF_B_FRAME	Bi-directional inter coded MBAFF frame. Not supported in this version of MPEG2 Encoder.
	IVIDEO_MBAFF_IDR_FRAME	Intra coded MBAFF frame that can be used for refreshing video content. Not supported in this version of MPEG2 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	Default output frame status Default value: 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.
IVIDEO_RateControlPreset	IVIDEO_NONE	No rate control is used
	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.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	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 MPEG2 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 MPEG2 Encoder.
	XDM_LE_32	32-bit little endian stream. Not supported in this version of MPEG2 Encoder.
XDM_ChromaFormat	XDM_CHROMA_NA	Chroma format not applicable. Encoder assumes IMPEG2VENC_YUV_420IUV
	XDM_YUV_420P	YUV 4:2:0 planar. Not supported in this version of MPEG2 Encoder.
	XDM_YUV_422P	YUV 4:2:2 planar. Not supported in this version of MPEG2 Encoder.
	XDM_YUV_422IBE	YUV 4:2:2 interleaved (big endian). Not supported in this version of MPEG2 Encoder.
	XDM_YUV_422ILE	YUV 4:2:2 interleaved (little endian). Not supported in this version of MPEG2 Encoder.
	XDM_YUV_444P	YUV 4:4:4 planar. Not supported in this version of MPEG2 Encoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_YUV_411P	YUV 4:1:1 planar. Not supported in this version of MPEG2 Encoder.
	XDM_GRAY	Gray format. Not supported in this version of MPEG2 Encoder.
	XDM_RGB	RGB color format. Not supported in this version of MPEG2 Encoder.
	XDM_YUV_420SP	YUV 420 semiplanar (Luma 1st plane, * CbCr interleaved 2nd plane)
	XDM_ARGB8888	Alpha plane Not supported in this version of MPEG2 Encoder
	XDM_RGB555	RGB 555 color format Not supported in this version of MPEG2 Encoder
	XDM_RGB565	RGB 556 color format Not supported in this version of MPEG2 Encoder
	XDM_YUV_444ILE	YUV 4:4:4 interleaved (little endian) Not supported in this version of MPEG2 Encoder
XDM_CmdId	XDM_GETSTATUS	Query algorithm instance to fill Status structure
	XDM_SETPARAMS	Set run-time dynamic parameters through the DynamicParams structure
	XDM_RESET	Reset the algorithm
	XDM_SETDEFAULT	Initialize all fields in DynamicParams structure to default values specified in the library
	XDM_FLUSH	Handle end of stream conditions. This command forces algorithm instance to output data without additional input. Not supported in this version of MPEG2 Encoder.
	XDM_GETVERSION	Query the algorithm version.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`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_SPEED` 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
	`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 MPEG2 Encoder Symbolic Constants and Enumerated Data Types

Table 4-2. List of Symbolic Constants

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
`IMP2VENC_RCAlgo`	`IMP2VENC_RC_NONE`	No rate control is used
	`IMP2VENC_RC_CBR`	Constant Bit-Rate (CBR) control for video conferencing.
	`IMP2VENC_RC_VBR`	Variable Bit-Rate (VBR) control for local storage (DVD) recording. This is the default value.
	`IMP2VENC_RC_DEFAULT`	Set to `IMP2VENC_RC_VBR`
`IMP2VENC_ChromaFormat`	`IMP2VENC_YUV_420IUV`	YUV420 format with UV interleaved. This is the only supported chroma format for this version of encoder.
`IMP2VENC_AspectRatio`	`IMP2VENC_AR_SQUARE`	1:1 Square See Table 6-3 in MPEG-2 visual standard.
	`IMP2VENC_AR_3_4`	3:4. See Table 6-3 in MPEG-2 visual standard.
	`IMP2VENC_AR_9_16`	9:16 See Table 6-3 in MPEG-2 visual standard.
	`IMP2VENC_AR_1_2_21`	1:2.21 See Table 6-3 in MPEG-2 visual standard. Not supported in the Main profile
	`IMP2VENC_AR_DEFAULT`	Set to `IMP2VENC_AR_SQUARE`

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
`MP2VENC_PixelRange`	`IMP2VENC_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-2 visual standard.

4.1.3 MPEG2 Encoder Error code Enumerated Data Types

Table 4-3. List of Error Codes

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IMP2VENC_STATUS	IMPEG2VENC_ERR_HANDLE_NULL	This fatal error is returned when input handle is NULL. If the handle is NULL in algFree or algInit call this error is returned to callee function. If the handle is NULL in a control call this error is returned in sStatus->videncStatus.extendedError and control call returns IVIDENC1_EFAIL. If the handle is NULL in a process call this error is returned in outArgs->extendedError and process call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_INCORRECT_HANDLE	This fatal error is returned when incorrect codec handle is passed to code API. If the handle is incorrectly passed in algFree or algInit call this error is returned to callee function. If the handle is incorrectly passed in a control call this error is returned in sStatus->videncStatus.extendedError and control call returns IVIDENC1_EFAIL. If the handle is incorrectly passed in a process call this error is returned in outArgs->extendedError and process call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_MEMTAB_NULL	This fatal error is returned when memtabs passed to algInit or algFree are NULL or not aligned to 32bit word boundary.
	IMPEG2VENC_ERR_IVIDENC1_INITPARAMS_SIZE	This fatal error is returned when size of algParams passed to algInit is not set to size of IVIDENC1_Params or size of IMPEG2VENC_Params.
	IMPEG2VENC_ERR_MEMTABS_SIZE	This fatal error is returned when size of memTabs passed to algInit are less than the requested size specified in algAlloc.
	IMPEG2VENC_ERR_MEMTABS_ATTRS	This fatal error is returned when attrs of memTabs passed to algInit are not as per the requested attrs specified in algAlloc.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`IMPEG2VENC_ERR_MEMTABS_SPACE`	This fatal error is returned when `space` of `memTabs` passed to `algInit` are not as per the requested `space` specified in `algAlloc`.
	`IMPEG2VENC_ERR_MEMTABS_BASE_NULL`	This fatal error is returned when `base` pointer of `memTabs` passed to `algInit` are `NULL`.
	`IMPEG2VENC_ERR_MEMTABS_BASE_NOT_ALIGNED`	This fatal error is returned when `base` pointer of `memTabs` passed to `algInit` are not aligned as per the requested alignment specified in `algAlloc`.
	`IMPEG2VENC_ERR_MEMTABS_OVERLAP`	This fatal error is returned if any two `memTabs` passed to `algInit` are overlapping in memory.
	`IMPEG2VENC_ERR_INV_CODEC_ID`	This fatal error is returned when invalid codec id is passed in the algProcess call
	`IMPEG2VENC_ERR_ENCODINGPRESET`	This fatal error is returned during `algInit` if the `encodingPreset` parameter is out of supported range `XDM_DEFAULT` (0) to `XDM_USER_DEFINED` (3) inclusive
	`IMPEG2VENC_ERR_INPUTCHROMAFORMAT`	This fatal error is returned during `algInit` if `inputChromaFormat` is not set to `XDM_YUV_420SP` or `XDM_CHROMA_NA`.
	`IMPEG2VENC_ERR_MAXFRAMERATE`	`maxFrameRate` not supported. “Fatal input error” is returned during `algInit` if `maxFrameRate` exceeds max supported value or is less than 0. The maximum supported value of frame rate is 60000 for high and high-1440 levels 30000 for low and main levels.
	`IMPEG2VENC_ERR_MAXBITRATE`	`maxBitRate` not supported. Fatal input error is returned during `algInit` if `maxBitRate` exceed max supported value as per the level or is less than 0. The maximum supported value of bit rate is 80 mbps for high level 60 mbps for high-1440 level 15 mbps for main level 4 mbps for low level

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG2VENC_ERR_MAXWIDTH	maxWidth not supported. "Fatal input error" is returned in algInit instance creation stage if maxWidth in the input params exceeds MPEG2VENC_TI_MAX_WIDTH (1920) or is less than MPEG2VENC_TI_MIN_WIDTH (128).
	IMPEG2VENC_ERR_MAXHEIGHT	maxHeight not supported. "Fatal input error" is returned in algInit instance creation stage if maxHeight in the input params exceeds MPEG2VENC_TI_MAX_HEIGHT (1088) or is less than MPEG2VENC_TI_MIN_HEIGHT (96).
	IMPEG2VENC_ERR_MAX_TOTAL_MBS	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if the number of MBs in a frame exceeds the maximum limit for resolution of 1920x1088. Control call returns IVIDENC1_EFAIL
	IMPEG2VENC_ERR_DATAENDIANNESS	dataEndianness not supported. fatal input error is returned during algInit if dataEndianness is not set to XDM_BYTE.
	IMPEG2VENC_ERR_MAXINTERFRAMEINTERVAL	This fatal unsupported param error is returned in algInit instance creation if maxInterFrameInterval not 0 or 1.
	IMPEG2VENC_ERR_RECONCHROMAFORMAT	reconChromaFormat not supported. fatal input error is returned during algInit if reconChromaFormat is not set to XDM_YUV_420SP or XDM_CHROMA_NA.
	IMPEG2VENC_ERR_INPUTCONTENTTYPE	inputContentType not supported. fatal input error is returned during algInit if inputContentType is not set to IVIDEO_PROGRESSIVE or IVIDEO_INTERLACED.
	IMPEG2VENC_ERR_RATECONTROLPRESET	rateControlPreset not supporteded fatal input error is returned during algInit if the rateControlPreset parameter is out of supported range 0 to IVIDEO_USER_DEFINED (5) inclusive.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG2VENC_ERR_TIMER_RESOLUTION	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if timerResolution is less than or equal to zero.
	IMPEG2VENC_ERR_ASPECTRATIO	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if aspectRatio is none of the IMP2VENC_AR_SQUARE, IMP2VENC_AR_3_4, IMP2VENC_AR_9_16 and IMP2VENC_AR_DEFAULT
	IMPEG2VENC_ERR_PIXELRANGE	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if pixelRange is none of the IMP2VENC_PR_16_235, IMP2VENC_PR_0_255 and IMP2VENC_PR_DEFAULT
	IMPEG2VENC_ERR_RESETHDVICPEVERYFRAME	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if resetHDVICPeveryFrame extended dynamic parameter is not 0 or 1. Control call returns IVIDENC1_EFAIL..
	IMPEG2VENC_ERR_METYPE	ME_Type not supported. fatal input error is returned during algInit if meAlgo is not 0 or 1.
	IMPEG2VENC_ERR_IVIDENC1_PROCESS_ARGS_NULL	This fatal error is returned in process call if any of input handle or inBufs or inArgs or outBufs are NULL.
	IMPEG2VENC_ERR_IVIDENC1_INARGS_SIZE	This fatal error can be retrieved from a XDM_GETSTATUS control call if inArgs size in process call was not set to IVIDENC1_inArgs or IMPEG2VENC_inArgs. Process call returns IVIDENC1_EFAIL.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG2VENC_ERR_IVIDENC1_OUTARGS_SIZE	This fatal error can be retrieved from a XDM_GETSTATUS control call if outArgs size in process call was not set to IVIDENC1_OutArgs or IMPEG2VENC_OutArgs. Process call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_IVIDENC1_INARGS_INPUTID	This fatal error is returned in outArgs->extendedError if inputID in inArgs of process call is 0. Process call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_IVIDENC1_INARGS_TOPFIELDFIRSTFLAG	This fatal error is retruned in outArgs->extendedError if topFieldFirstFlag in inArgs is not set correctly to XDAS_TRUE for interlace content. Process call returns IVIDENC1_EFAIL. Currently only XDAS_TRUE is supported
	IMPEG2VENC_ERR_IVIDENC1_INBUFS	This fatal error is returned in outArgs->extendedError if inBufs is null or if numBufs in inBufs is not set to 2 or if frameWidth and frameHeight in inBufs are not equal to inputWidth and inputHeight of XDM_SETPARAMS control call. Process call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_IVIDENC1_INBUFS_BUFDESC	This fatal error is returned in outArgs->extendedError if buffer descriptors in inBufs are either NULL or if their sizes are less than the frame size. Process call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_IVIDENC1_OUTBUFS	This fatal error is returned in outArgs->extendedError if outBufs is NULL or if numBufs in outBufs is less than 1 when mvSADoutFlag=0 or if numBufs in outBufs is less than 2 when mvSADoutFlag=1. Process call returns IVIDENC1_EFAIL.

Group or Enumeration Class	**Symbolic Constant Name**	**Description or Evaluation**
	IMPEG2VENC_ERR_IVIDENC1_OUTBUFS_NULL	This fatal error is returned in `outArgs->extendedError` if `bufs` or `bufSizes` of `outBufs` is `NULL`, Process call returns `IVIDENC1_EFAIL`.
	IMPEG2VENC_ERR_CODEC_INACTIVE	This fatal error is returned in `outArgs->extendedError` if the codec is not activated by means of algActivate call before invoking a process call.
	IMPEG2VENC_ERR_CODEC_NOT_INITIALIZED	This fatal error is returned in `outArgs->extendedError` if the resource initialization for codec is not properly done using IRES functions
	IMPEG2VENC_ERR_INPUTWIDTH_NON_MULT_OF_16	`inputWidth` not supported. Fatal input error is returned in `videncStatus.extendedError` during XDM_SETPARAMS control call if the `inputWidth` in input dynamic params is a non-multiple of 16. Control call returns `IVIDENC1_EFAIL`
	IMPEG2VENC_ERR_INPUTHEIGHT_NON_MULT_OF_16	`inputHeight` not supported. Fatal input error is returned in `videncStatus.extendedError` during XDM_SETPARAMS control call if the `inputHeight` in input dynamic params is a non-multiple of 16. Control call returns `IVIDENC1_EFAIL`

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG2VENC_ERR_INPUTWIDTHHEIGHT	inputWidth not supported fatal input error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if the inputWidth in input dynamic params exceeds maxWidth or if inputWidth is less than MPEG2VENC_TI_MIN_WIDTH(64). Control call returns IVIDENC1_EFAIL inputHeight not supported fatal input error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if the inputHeight in input dynamic params exceeds maxHeight or if inputHeight is less than MPEG2VENC_TI_MIN_HEIGHT(64). Control call returns IVIDENC1_EFAIL
	IMPEG2VENC_ERR_CAPTUREWIDTH	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if captureWidth is not 0 and less than inputWidth. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_GENERATEHEADER	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if generateHeader is not 0 or 1. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_INTERFRAMEINTERVAL	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if interFrameInterval is not 0 or 1. Control call returns IVIDENC1_EFAIL.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG2VENC_ERR_BITRATE	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if targetBitRate in dynamic params exceeds maxBitRate or less than 0. Control call returns IVIDENC1_EFAIL. The maximum supported value of bit rate is 80 mbps for high level 60 mbps for high-1440 level 15 mbps for main level 4 mbps for low level
	IMPEG2VENC_ERR_REFFRAMERATE_MISMATCH	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if refFrameRate and targetFrameRate mismatch. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_TARGETFRAMERATE	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if targetFrameRate in dynamic params exceeds maxFrameRate or is less than 0. Control call returns IVIDENC1_EFAIL. The supported values are ❑ 24000, 25000, 30000 for low and main levels ❑ 24000, 25000, 30000, 50000 and 60000 for high and high-1440 levels
	IMPEG2VENC_ERR_INTRAFRAMEINTERVAL	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if intraFrameInterval is less than 0. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_FORCEFRAME	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if forceFrame is not IVIDEO_NA_FRAME or IVIDEO_I_FRAME. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_TIMESCALE	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if ((TimerResolution * 1000) is less than targetFrameRate.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG2VENC_ERR_INTERFRAMEQP	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if qpInter is less than 1 or more than 31. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_INTRAFRAMEQP	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if qpIntra is less than 1 or more than 31. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_INITQ	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if qpInit is less than 1 or more than 31. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_QPMAX	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if QPMax is less than 1 or more than 31. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_QPMIN	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if QPMin is less than 1 or more than 31. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_QPMIN_EXCEEDS_QPMAX	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if QPMin is greater than QPMax. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_RCALGO	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if RcAlgo is not set to 0, 4 or 8 when rateControlPreset is IVIDEO_USER_DEFINED. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_MAXDELAY	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if maxDelay is less than 100 or greater than 10000. Control call returns IVIDENC1_EFAIL.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG2VENC_ERR_PERCEPTUALRC	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if perceptualRC is not 0. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_QSCALETYPE	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if QscaleType is not 0 or 1. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_INTRA_DC_PREC	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if IntraDCPrec is less than zero or greater than three. Control call returns IVIDENC1_EFAIL.
	IMPEG2VENC_ERR_MV_SAD_OUT_FLAG	This fatal error is returned in videncStatus.extendedError during XDM_SETPARAMS control call if mvSADoutFlag is not 0 or 1. Control call returns IVIDENC1_EFAIL

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`

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	If the buffer was not accessed by the algorithm processor (for example, it was filled through DMA or other hardware accelerator that does not write through the algorithm CPU), then bits in this mask should not be set.

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 MPEG2 Main 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 when mvSADoutFlag=0 and 2 when mvSADoutFlag=1.
- ❑ The input buffer sizes (in bytes) for worst case 1920x1088 in

YUV420SP format are:

Y buffer = 1920 x 1088
UV buffer = 1920 x 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.

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

- ❑ There is no restriction on output buffer size except that it should be enough to store one frame of encoded data.The output buffer size returned by the `XDM_GETBUFINFO` command assumes that the worst case output buffer size is maximum of ((1000*1024) and (`frameHeight*frameWidth`)/2.

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

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 enumerated datatype `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 and

Field	Datatype	Input/ Output	Description
			updated in this version of MPEG2 Encoder on DM365.
`repeatFirstFieldFlag`	`XDAS_Int32`	Output	Flag to indicate when the first field should be repeated Note: This feature is not supported and updated in this version of MPEG2 Encoder on DM365.
`frameStatus`	`XDAS_Int32`	Output	Status of the output frame This takes one of the values from enumerated datatype `IVIDEO_OutputFrameStatus` as described in Table 4-1.
`repeatFrame`	`XDAS_Int32`	Output	Number of times the display process should repeat.
`contentType`	`XDAS_Int32`	Output	The output data content type. This takes one of the values from enumerated datatype `IVIDEO_ContentType` as described in Table 4-1.
`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. Note: It will be same as `inputWidth` for width multiple of 16. For `inputWidth` non-multiple of 16, application will set this field to next multiple of 16.
`frameHeight`	`XDAS_Int32`	Input	Height of the video frame. Note: Progressive: It will be same as `inputHeight` for height multiple of 16.For `inputHeight` non-multiple of 16, application will set this field to next multiple of 16. Interlaced: It will be same as `inputHeight` for height multiple of 32.For `inputHeight` non-multiple of 32, application will set this field to next multiple of 32.
`framePitch`	`XDAS_Int32`	Input	Frame pitch used to store the frame. This field is not used by the encoder.
`bufDesc[XDM_MAX_IO_BUFFERS]`	`XDM1_SingleBufDesc`	Input	Picture buffers

4.2.1.8 IVIDENC1_Fxns

‖ Description

This structure contains pointers to all the XDAIS and XDM interface functions.

‖ Fields

Field	Data type	Input/ Output	Description
`ialg`	`IALG_Fxns`	Input	Structure containing pointers to all the XDAIS interface functions. For more details, see *TMS320 DSP Algorithm*

Field	Data type	Input/ Output	Description
			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 `IVIDENC1_Params` structure.
`encodingPreset`	`XDAS_Int32`	Input	Encoding preset. See `XDM_EncodingPreset` enumeration for details. Default value = `XDM_DEFAULT`.
`rateControlPreset`	`XDAS_Int32`	Input	Rate control preset. See `IVIDEO_RateControlPreset` enumeration for details. Default value = `IVIDEO_LOW_DELAY`.
`maxHeight`	`XDAS_Int32`	Input	Maximum video height to be supported in pixels. Maximum height supported in this version is 1088. Minimum height supported is 96. Default value = 1088
`maxWidth`	`XDAS_Int32`	Input	Maximum video width to be supported in pixels. Maximum width supported in this version is 1920. Minimum width supported is 128. Default value = 1920.
`maxFrameRate`	`XDAS_Int32`	Input	Maximum frame rate in fps * 1000 to be supported. Default value = 60000.
`maxBitRate`	`XDAS_Int32`	Input	Maximum bit-rate to be supported in bits per second. Default value = 80Mbps.

Field	Data type	Input/ Output	Description
dataEndianness	XDAS_Int32	Input	Endianness of input data. See XDM_DataFormat enumeration for details. Default value = XDM_BYTE.
maxInterFrameInterval	XDAS_Int32	Input	Distance from I-frame to P-frame: ❑ 1 - If no B-frames ❑ 2 - To insert one B-frame This parameter is not supported as B-frames are not supported. Set value = 1
inputChromaFormat	XDAS_Int32	Input	Input chroma format. See XDM_ChromaFormat and IMPEG2VENC_ChromaFormat enumeration for details. Set value as = XDM_YUV_420SP. Other values are not supported.
inputContentType	XDAS_Int32	Input	Input content type. See IVIDEO_ContentType enumeration for details. Default value = IVIDEO_PROGRESSIVE. Only field picture encoding is supported in the case of inputContentType = IVIDEO_INTERLACE. Application has to invoke two process calls for each field with first process call having top field data and second process call having bottom field data. The bit stream will be available after every field encode however the reconstructed picture will be given out only after both the process calls are executed.
reconChromaFormat	XDAS_Int32	Input	Chroma formats for the reconstruction buffers. Set value as = XDM_YUV_420SP. Other values are not supported.

Note:

The maximum video height and width supported are 1920 and 1088 pixels respectively.

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

The following fields of IVIDENC1_Params data structure are level dependent:

❑ maxHeight

❑ maxWidth

❑ maxFrameRate

❑ maxBitRate

4.2.1.10 IVIDENC1_DynamicParams

‖ Description

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

‖ Fields

Field	Data type	Input/ Output	Description
`size`	`XDAS_Int32`	Input	Size of the basic or extended (if being used) data structure in bytes. Default value is size of `IVIDENC1_DynamicParams` structure.
`inputHeight`	`XDAS_Int32`	Input	Height of input frame in pixels. Input height can be changed before start of encoding within the limits of maximum height set in creation phase. `inputHeight` must be multiple of two. Minimum height supported is 96. Irrespective of interlaced or progressive content, input height should be given as frame height. Note: Progressive: When the input height is a non-multiple of 16, the encoder expects the application to pad the input frame to the nearest multiple of 16 at the bottom of the frame. In this case, the application should set input height to actual height but should provide the padded input YUV data buffer to encoder. Interlaced: When the input height is a non-multiple of 32, the encoder expects the application to pad the input frame to the nearest multiple of 32 at the bottom of the frame. In this case, the application should set input height to actual height but should provide the padded input YUV data buffer to encoder. Default value = 1088.

Field	Data type	Input/ Output	Description
`inputWidth`	`XDAS_Int32`	Input	Width of input frame in pixels. Input width can be changed before the start of encoding within the limits of maximum width set in creation phase. `inputWidth` must be multiples of two. Minimum width supported is 128. Note: When the input width is a non-multiple of 16, the encoder expects the application to pad the input frame to the nearest multiple of 16 to the right of the frame. In this case, application should set `inputWidth` to actual width but should provide the padded input YUV data buffer to encoder.. Default value = 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 = 60000
`targetFrameRate`	`XDAS_Int32`	Input	Target frame rate in fps * 1000. For example, if the frame rate is 30, set this field to 30000. Default value = 60000.
`targetBitRate`	`XDAS_Int32`	Input	Target bit-rate in bits per second. For example, if the bit-rate is 2 Mbps, set this field to 2000000. Default value = 4000000.
`intraFrameInterval`	`XDAS_Int32`	Input	Interval between two consecutive intra frames. ❑ 0: First frame will be intra coded ❑ 1: No inter frames, all intra frames ❑ 2: Consecutive IPIPIP ❑ 3: 1PPIPPIPP or IPBIPBIPB, and so on Default value = 15
`generateHeader`	`XDAS_Int32`	Input	Encode entire access unit or only header. See `XDM_EncMode` enumeration for details. Default value = `XDM_ENCODE_AU`.

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

Note:

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

❑ `inputHeight <= maxHeight`

❑ `inputWidth <= maxWidth`

❑ `refFrameRate <= maxFrameRate`

❑ `targetFrameRate <= maxFrameRate`

❑ `targetFrameRate` should be multiple of 1000

❑ The value of the `refFrameRate` and `targetFrameRate` should be the same.

- `targetBitRate <= maxBitRate`
- The `inputHeight` and `inputWidth` must be multiples of two.
- The `inputHeight`, `inputWidth`, and `targetFrameRate` should adhere to the standard defined level limits. As per the requirement, level limit can be violated for `targetBitRate`.
- When `inputHeight`/`inputWidth` are non-multiples of 16, encoder expects the application to pad the input frame to the nearest multiple of 16 at the bottom/right of the frame. In this case, application sets the `inputHeight`/`inputWidth` to the `actual height/actual width`; however, it should provide the padded input YUV data buffer to the encoder.
- When `inputWidth` is non-multiple of 16, the encoder expects capture width as padded width(nearest multiple of 16). If the capture width is 0 or less than padded width, then the capture width is assumed to be the padded width. In all other cases, the capture width provided through input parameter is used for input frame processing.
- For out of bound and invalid parameters, encoder returns with fatal error.

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.

Field	Data type	Input/ Output	Description
			This field is only applicable to the input image buffer. Currently, supported values are: ❑ XDAS_TRUE for interlace encoding ❑ XDAS_TRUE and XDAS_FALSE for progressive encoding. In case of interlaced encoding, the first field to be encoded is always assumed to be the top field. Hence, the flag value `XDAS_FALSE` is not supported. The bit-stream output will be available immediately after each field encode. However, the reconstructed output will be available only after the encode of both the fields

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 4.1.1 and codec specific code in section 4.1.3 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)

Field	Data type	Input/ Output	Description
			data structure in bytes.
`extendedError`	`XDAS_Int32`	Output	Extended error code. See `XDM_ErrorBit` enumeration in section 4.1.1 and codec specific code in section 4.1.3 for details.
`bytesGenerated`	`XDAS_Int32`	Output	The number of bytes generated.
`encodedFrameType`	`XDAS_Int32`	Output	Frame types for video. See `IVIDEO_FrameType` enumeration for details. Following values are only supported ❑ `IVIDEO_I_FRAME` ❑ `IVIDEO_P_FRAME` ❑ `IVIDEO_II_FRAME` ❑ `IVIDEO_PP_FRAME` ❑ `IVIDEO_IP_FRAME` ❑ `IVIDEO_PI_FRAME`
`inputFrameSkip`	`XDAS_Int32`	Output	Frame skipping modes for video. See `IVIDEO_SkipMode` enumeration for details.
`outputID`	`XDAS_Int32`	Output	Output ID corresponding to the encoder buffer. This can also be used to free the corresponding image buffer for further use by the client application code. In this encoder, outputID is set to `IVIDENC1_InArgs::inputID.`
`encodedBuf`	`XDM1_SingleBufDesc`	Output	The encoder fills the buffer with the encoded bit-stream. In case of sequences with only I and P frames, these values are identical to `outBufs` passed in `IVIDENC1_Fxns::process()` The `encodedBuf.bufSize` field returned corresponds to the actual valid bytes available in the buffer. The bit-stream is in encoded order. The `outputId` and `encodedBuf` together provide information related to the corresponding encoded image buffer.
`reconBufs`	`IVIDEO1_BufDesc`	Output	Pointer to reconstruction buffer descriptor.

4.2.2 MPEG2 Encoder Data Structures

This section includes the following MPEG2 Encoder specific extended data structures:

- `IMPEG2VENC_Params`
- `IMPEG2VENC_DynamicParams`
- `IMPEG2VENC_InArgs`
- `IMPEG2VENC_Status`
- `IMPEG2VENC_OutArgs`
- `IMPEG2VENC_Fxns`

4.2.2.1 IMPEG2VENC_Params

‖ Description

This structure defines the creation parameters and any other implementation specific parameters for a MPEG2 Encoder instance object. The creation parameters are defined in the XDM data structure, `IVIDENC1_Params`.

‖ Fields

Field	Data type	Input/ Output	Description
`videncParams`	`IVIDENC1_Params`	Input	See `IVIDENC1_Params` data structure for details. The size parameter in `videncParams` is set to size of `IMPEG2VENC_Params` structure by default while using extended parameters.
`aspectRatio`	`XDAS_Int32`	Input	Aspect ratio to be put in header See `IMP2VENC_AspectRatio` enumeration for details. Default value is `IMP2VENC_AR_DEFAULT`.
`pixelRange`	`XDAS_Int32`	Input	Pixel range to be put in header See `IMP2VENC_PixelRange` enumeration for details. Default value is `IMP2VENC_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 60.

Field	Data type	Input/ Output	Description
`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 1. Note: Use Low power search algorithm for better performance and Normal search algorithm for better quality.
`QscaleType`	`XDAS_Int32`	Input	Q Scale Type used by the encoder. It can take following values: ❑ 0 – Q Scale Type 0 ❑ 1 – Q Scale Type 1 Default value is 0.
`IntraDCPrec`	`XDAS_Int32`	Input	Precision of intra DC coefficient. It can take following values: ❑ 0 – 8 bits ❑ 1 – 9 bits ❑ 2 – 10 bits Default value is 0.
`hdvicpHandle`	`void`	Input	Pointer to the HDVICP handle structure. Note: Currently this parameter is not reserved and should be initialized to NULL.

Note:

Default values of extended parameters are used when size fields are set to the size of base structure `IVIDENC1_Params`.

4.2.2.2 IMPEG2VENC_DynamicParams

║ Description

This structure defines the run-time parameters and any other implementation specific parameters for a MPEG2 Encoder instance object. The run-time parameters are defined in the XDM data structure, `IVIDENC1_DynamicParams`.

║ Fields

Field	Data type	Input/ Output	Description
`videncDynamicParams`	`IVIDENC1_DynamicParams`	Input	See `IVIDENC1_DynamicParams` data structure for details. The size parameter of `DynamicParams` is set to size of `IVIDENC1_DynamicParams` structure by default while using extended parameters.
`qpIntra`	`XDAS_Int32`	Input	Default Quantization Parameter (QP) for I frame. This value must be between 1 and 31 for MPEG-2. By default, this is set to 8 at the time of encoder object creation.
`qpInter`	`XDAS_Int32`	Input	Quantization parameter for P frame. This value must be between 1 and 31 for MPEG-2. By default, this is set to 8 at the time of encoder object creation.
`RcAlgo`	`XDAS_Int32`	Input	Rate control algorithm to be used. See `IMP2VENC_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 `IMP2VENC_RC_DEFAULT`. Note: All the supported RC algorithms do not support quantization scale variation within the I-frames either at row level or at MB level.
`QPMax`	`XDAS_Int32`	Input	Max Quantization Parameter (QP) to be used by encoder. This value must be between 1 and 31 for MPEG-2. This should be greater than QPMin. By default, this is set to 31 at the time of encoder object creation.

Field	Data type	Input/ Output	Description
`QPMin`	`XDAS_Int32`	Input	Max Quantization Parameter (QP) to be used by encoder. This value must be between 1 and 31 for MPEG-2. This should be less than `QPMax`. By default, this is set to 1 at the time of encoder object creation.
`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 `IMP2VENC_RC_NONE` and bit-rate is a non-zero value. This value must be between 1 and 31 for MPEG-2. 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. By default, this is set to 8 at the time of encoder object creation.
`PerceptualRC`	`XDAS_Int32`	Input	Flag to indicate that the encoder should enable perceptual rate control option while encoding. ❑ 1 - Enable Perceptual RC ❑ 0 – Disable Perseptual RC This option will be used when `RcAlgo` is set to either `IMP2VENC_RC_VBR or IMP2VENC_RC_CBR` Note: This field is currently a reserved field.

Field	Data type	Input/ Output	Description
`reset_vIMCOP_every_frame`	`XDAS_Int32`	Input	Flag to reset HDVICP at the start of every frame being encoded. This is useful for multi-channel encoding of different HDVICP video codecs. ❑ 1 - ON ❑ 0 - OFF Default value is 1. If this flag is set, MPEG2 encoder assumes that the memory of HDVICP was overwritten by some other codec instance 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 MPEG2 decoder. However, application can set this flag to 0 for better performance if it runs multiple instances of MPEG2 encoder alone.
`mvSADoutFlag`	`XDAS_Int32`	Input	Flag to enable exporting of ME info like ME Sad, Motion vectors through output buffers ❑ 1 - ON ❑ 0 - OFF Default value is 0.

Note:

- `RcAlgo` values are used only when `IVIDENC1_Params->RateControlPreset = IVIDEO_USER_DEFINED`.
- `QPMax`, `QPMin`, `qpInit`, and `maxDelay` values are used only when the encoder does not run in fixed QP mode.
- The MV and SAD is dumped in the `outBuf`. The extra buffer is requested during `XDM_GETBUFINFO` callthe MV-SAD is dumped in index 1 of buffer pointers. Index 0 is always used for bit-stream data. MV SAD information is in the following format:
  - Word0: SAD [bit 31-0]
  - Word1: MVx[bit 31-16]:MVy[bit 15-0]

4.2.2.3 IMPEG2VENC_InArgs

‖ Description

This structure defines the run-time input arguments for MPEG2 Encoder instance object.

‖ Fields

Field	Data type	Input/ Output	Description
`videncInArgs`	`IVIDENC1_InArgs`	Input	See `IVIDENC1_InArgs` data structure for details.
`TimeStamp`	`XDAS_Int32`	Input	Time stamp value of the frame to be placed in bit stream. This should be integral multiple of `TimerResolution`/ (frame rate in fps). Initial time stamp value (for first frame) should be 0. Default is calculated as Frame number * `TimerResolution`/ (Frame rate in fps). See Appendix A for more details.

4.2.2.4 IMPEG2VENC_Status

‖ Description

This structure defines parameters that describe the status of the MPEG2 Encoder and any other implementation specific parameters. The status parameters are defined in the XDM data structure, `IVIDENC1_Status`.

‖ Fields

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

4.2.2.5 IMPEG2VENC_OutArgs

‖ Description

This structure defines the run-time output arguments for the MPEG2 Encoder instance object.

‖ Fields

Field	Data type	Input/ Output	Description
`videncOutArgs`	`IVIDENC1_OutArgs`	Output	See `IVIDENC1_OutArgs` data structure for details.

4.2.2.6 IMPEG2VENC_Fxns

‖ Description

This structure defines all of the operations for the MPEG2 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 MPEG2 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:

The control API can be called with base or extended `DynamicParams`, and `Status` data structure. If you are using extended data structures, the third and fourth arguments must be pointers to the extended `DynamicParams` and `Status` data structures respectively. Also, ensure that the `size` field is set to the size of the extended data structure. Depending on the value set for the `size` field, the algorithm uses either basic or extended parameters.

‖ Preconditions

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

- ❑ `control()` can only be called after a successful return from `algInit()` and `algActivate()`.
- ❑ `handle` must be a valid handle for the algorithm's instance object.

‖ Post conditions

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

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

‖ Example

See test application file, mpeg2encoderapp.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, mpeg2encoderapp.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 MPEG2 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

Time-Stamp Insertion

A.1 Description

The DM365 MPEG2 Encoder supports insertion of frame time-stamp. The time-stamp is useful for audio-synchronization and determining the exact timing for display of frames. The application should take proper care while setting the parameters for time-stamp and the actual time-stamp for each frame. Ideally, the time-stamp can be set based on the frame-rate. This simplifies the process of generating time-stamps. However, the application is free to use any method of time-stamp generation.

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

```
T = (R * 1000) / F

where

F: targetFrameRate

R: timerResolution

T: units_per_frame
```

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

Example 1.

```
F = 30000.

Let R = 60

T = (60 * 1000) / 30000

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

Example 2.

```
F = 25000
R = 25
T = (25 * 1000) / 25000
```

TimeStamp = 0, 1, 2, 3, 4...

Appendix B

Revision History

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

Table B-1. Revision History for MPEG2 MP Encoder on DM365

Section	Addition/Deletion/Modifications
Global changes	❑ There are no changes in the user guide for this relases of MPEG2 MP Encoder on DM365