MPEG4 Simple Profile Encoder Codec on DM355

User's Guide

Literature Number: SPRUFE4C
October 2007–Revised April 2008

Contents

List of Figures

List of Tables

Read This First

This document describes how to install and work with Texas Instruments' (TI) MPEG4 Simple Profile Encoder implementation on the DM355 platform. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

About This Manual

TI codec implementations are based on the eXpressDSP Digital Media (xDM) standard. XDM is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS).

Intended Audience

This document is intended for system engineers who want to integrate TI codecs with other software to build a multimedia system based on the DM355 platform.

This document assumes that you are fluent in the C language, and have working knowledge of MPEG4 encoder. Good knowledge in eXpressDSP Algorithm Interface Standard (XDAIS), eXpressDSP Digital Media (xDM) standard, and IDMA3 will be helpful.

How to Use This Manual

This document includes the following chapters:

- **Chapter 1 - Introduction**, provides a brief introduction to the XDAIS and XDM standards. It also provides an overview of the codec and lists its supported features.
- **Chapter 2 - Installation Overview**, describes how to install, build, and run the codec.
- **Chapter 3 - Sample Usage**, describes the sample usage of the codec.
- **Chapter 4 - API Reference**, describes the data structures and interface functions used in the codec.

Related Documentation From Texas Instruments

The following documents describe TI 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 API Reference* (SPRU360) describes all the APIs that are defined by the TMS320 DSP Algorithm Interface Standard (also known as XDAIS) specification.
- *A Technical Overview of eXpressDSP - Compliant Algorithms for DSP Software Producers* (SPRA579) describes how to make algorithms compliant with the TMS320 DSP Algorithm Standard which is part of TI eXpressDSP technology initiative.
- *xDAIS-DM (Digital Media) User Guide* (SPRUEC8)

2.2.1 Related Documentation

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

- *ISO/IEC 14496-2:2004, Information technology -- Coding of audio-visual objects -- Part 2: Visual* (Approved in 2004-05-24)
- *H.263 ITU-T Standard – Video Coding for low bit rate communication*

2.2.2 Abbreviations

The following abbreviations are used in this document.

Table 1. List of Abbreviations

Abbreviation	Description
CBR	Constant Bit Rate
CIF	Common Intermediate Format
CSL	Chip Support Library
DCT	Discrete Cosine Transform
DMA	Direct Memory Access
DMAN3	DMA Manager
EVM	Evaluation Module
GOB	Group of Blocks
GOV	Group of VOP
HEC	Header Extension Code
HPI	Half Pel Interpolation
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ITU	International Telecommunications Union
MPEG	Moving Pictures Experts Group
QCIF	Quarter Common Intermediate Format
QP	Quantization Parameter
QVGA	Quarter Video Graphics Array
SP	Simple Profile
VBR	Variable Bit Rate
CVBR	Constrained Variable Bit Rate
VBV	Video rate Buffer Verifier
VOL	Video Object Layer
VOP	Video Object Plane
VOS	Video Object Sequence
UMV	Unrestricted Motion Vector
XDAIS	eXpressDSP Algorithm Interface Standard
XDM	eXpressDSP Digital Media
IDMA3	DMA Resource specification and negotiation protocol

2.2.2.1 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

2.2.2.2 Product Support

When contacting TI for support on this codec, please quote the product name (MPEG4 Simple Profile Encoder on DM355) and version number. The version number of the codec is included in the Title of the Release Notes that accompanies this codec.

2.2.2.2.1 Trademarks

Code Composer Studio and eXpressDSP are trademarks of Texas Instruments.

All trademarks are the property of their respective owners.

2.2.2.2.2 Software Copyright

Introduction

This chapter provides a brief introduction to XDAIS and xDM. It also provides an overview of TI implementation of the MPEG4 Simple Profile Encoder on the DM355 platform and its supported features.

Topic **Page**

1.1 Overview of XDAIS, XDM and IDMA3

TI multimedia codec implementations are based on the eXpressDSP Digital Media (xDM) standard. XDM is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS). IDMA3 is the standard interface to algorithms for DMA resource specification and negotiation protocols. This interface allows the client application to query and provide the algorithm with its requested DMA resources.

1.1.1 XDAIS Overview

An eXpressDSP-compliant algorithm is a module that implements the abstract interface IALG. The IALG API takes the memory management function away from the algorithm and places it in the hosting framework. Thus, an interaction occurs between the algorithm and the framework. This interaction allows the client application to allocate memory for the algorithm and also share memory between algorithms. It also allows the memory to be moved around while an algorithm is operating in the system. 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 process data in real-time. The algActivate() API provides a notification to the algorithm instance that one or more algorithm processing methods are about to be run zero or more times in succession. After the processing methods have been run, the client application calls the algDeactivate() API prior to reusing any of the instance's scratch memory.

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

1.1.2 XDM Overview

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

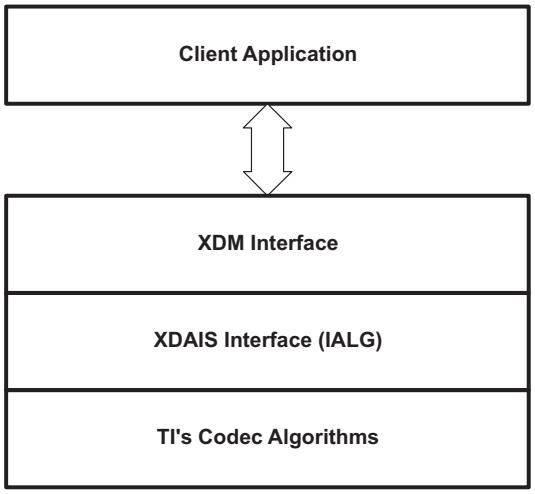
- control()
- process()

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

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

Figure 1-1 shows the XDM interface to the client application.

Figure 1-1. XDM Interface to the Client Application



As shown in the Figure 1-1, 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 multimedia algorithms are XDM compliant, it provides you with the flexibility to use any TI algorithm without changing the client application code. For example, if you have developed a client application using an XDM-compliant MPEG4 video encoder, then you can easily replace MPEG4 with another XDM-compliant video encoder with minimal changes to the client application.

For more details, see *xDAIS-DM (Digital Media) User Guide* (SPRUEC8).

1.1.3 IDMA3 Overview

Client applications use the algorithm's IDMA3 interface to query the algorithm's DMA resource requirements and grant the algorithm logical DMA resources via handles. The algorithm specifies the number of separate EDMA/QDMA channels and PaRamsets it requires through memRecs. The IDMA3 standard defines following APIs:

- dmaChangeChannels()
- dmaGetChannelCnt()
- dmaGetChannels()
- dmaInit()

dmaChangeChannels( ) is called by an application whenever logical channels are moved at run-time. This allows for the application to re-initialize the channel properties whenever allocated resources are not available. dmaGetChannelCnt() is called by an application to query an algorithm about its number of logical DMA channel requests. dmaGetChannels() is called by an application to query an algorithm about its DMA channel requests at initialization time, or to get the current channel holdings. Through this API, the algorithm specifies the number of TCCs and PaRamSets it requires and the properties of these resources when called during initialization time. dmaInit() is called by an application to grant DMA handle(s) to the algorithm at initialization.

For more details, see *Using DMA with Framework Components for 'C64x+* (SPRAAG1).

1.2 Overview of MPEG4 Simple Profile Encoder

MPEG4 is the ISO/IEC recommended standard for video compression.

Refer to *ISO/IEC 14496-2:2004, Information technology -- Coding of audio-visual objects -- Part 2: Visual* (Approved in 2004-05-24) for details on MPEG4 encoding process.

All references in this document to the MPEG4 Encoder refer to the MPEG4 Simple Profile Encoder only.

1.3 Supported Services and Features

This user guide accompanies the TI implementation of the MPEG4 Encoder on the DM355 platform.

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

- eXpressDSP™ Algorithm Interface Standard (XDIAS) software compliant
- eXpressDSP Digital Media (xDM) interface and IDMA3 compliant
- Implements IVIDENC1 interface of xDM
- Compliant with the MPEG4 simple profile levels 0, 1, 2, 3. In addition, it can encode 720P (1280x720) and SXVGA (1280x960) formats.
- Supports YUV 4:2:2 interleaved data as an input
- Supports image width as multiple of 16 and height as multiple of 16
- Supports Half Pel Interpolation (HPI) for motion estimation
- Supports 1 motion vector encoding for motion estimation (1MV/MB) with (-32, +31) half pel search range
- Supports following motion estimation algorithms based on MEAlgo API parameter
  - DM355_MPEG4E_ME_LQ_HP (low quality and high performance)
  - DM355_MPEG4E_ME_MQ_MP (medium quality and medium performance)
  - DM355_MPEG4E_ME_HQ_MP (good quality and medium performance)
  - DM355_MPEG4E_ME_HQ_LP (best quality and low performance)
- Supports DC prediction. Supports AC prediction for VBR rate control mode with fixed Qp (rateFix = 1, IVIDEO_NONE and IMP4VENC_VBR_RATEFIX).
- Supports generation of streams with resync marker (RM)
- Supports MPEG2 Step 2 TM5 rate control algorithm
- Supports Constrained Variable Bit Rate (IVIDEO_STORAGE), Constant Bit Rate (IVIDEO_LOW_DELAY), Variable Bit Rate (IVIDEO_NONE) and Variable Bit Rate with rate fix range control (IMP4VENC_VBR_RATEFIX).
- Supports Intra – Inter decision at 16x16 level (for better speed) or 8x8 block level (for better quality) level based on IIDC API parameter
- Supports Bonus Skip MB logic (for better quality) or non-Bonus Skip MB logic (for better performance) based on SkipMBAlgo API parameter
- Supports Unrestricted Motion Vectors (UMV)
- Supports access of motion vectors and SAD through MV access API. The application should pass the buffer required to write the SAD and motion vector generated. This should be passed as an Output buffer parameter.
- Supports the VOL header generation at frame level. The application has to pass the buffer required to write the VOL header generated. The encoding process is bypassed and frame count is unaltered when the Header generation API is called.
- Supports modification of target bit rate and frame rate (including non-integer)
- Supports setting of separate Quantization Parameter (Qp) for I-frames and P-frames
- Supports changing the size of video packets at create time
- Supports Algorithm 0 (for better performance) and algorithm 1 (for better quality) for INTRA/INTER decision in P frames
- Supports Area Encode feature. The application can provide width, height, sub window width and sub window height to the algorithm for encoding. The sub-window width and sub-window height should be multiple of 16.
- Supports Rotation (0, 90, 180 and 270 degrees) integrated with the Encoder up to a resolution of 720x576.
- Supports changing the encoding parameters at runtime (Dynamic configurability)
- Supports frame level re-entrancy
- Supports multi instance of MPEG4 Encoder, and single/multi instance of MPEG4 Encoder with other DM355 codecs
- Validated on DM355 EVM

1.4 Limitations

The limitations will not be removed in future releases. These limitations are not defects but intentional or known deficiencies.

- Does not support 4 MV
- Does not support AC prediction for varying Qp (rateFix = 0)
- Does not support ME range beyond -32 and +31. Only ME Range = 31 and ME Range = 7 are supported.
- Does not support DP, RVLC and HEC
- Does not support any chroma format other than 422ILE for input
- Does not support input width/height, sub-window width/height, Rate control algorithm, VBV size, or rotation as dynamically configurable parameters.
- Does not support the combination of the bonus SKIP MB and intra Algo 'DM355_MPEG4E_INTRA_INTER_DECISION_LQ_HP'.
- Does not support motion vector access with intra Algo 'DM355_MPEG4E_INTRA_INTER_DECISION_LQ_HP'.
- Does not support arbitrary width and height
  - Supports image width as multiple of 16 and height as multiple of 16
  - Does not support image width below 160 (without UMV)
  - Does not support image width below 192 (with UMV).
- Rotation with width more than 720 or height more than 576 (for instance, 720p (1280x720) or SXVGA (1280x960)) is not supported
- Achieved bit rate varies within +/-10% of target bit rate in CBR (Constant Bit Rate) mode
- Minimum value of quantization parameter (Qp) is limited to 2; i.e., Qp=1 is not supported
- MV access API always provides the motion vectors for the best matching MB
- For Variable Bit Rate with rate fix range control (IMP4VENC_VBR_RATEFIX), maximum frame rate supported is 100.

Installation Overview

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

Topic **Page**

2.1 System Requirements

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

2.1.1 Hardware

This codec has been tested as an executable on DM355 board.

2.1.2 Software

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

- **Linux**: Monta Vista Linux 4.0.1
- **Code Generation Tools**: This project is compiled, assembled, and linked using the arm_v5t_le-gcc compiler.

2.2 Installing the Component

To install the codec, follow the instructions in the Release notes. The code location is as follows:

The MPEG4 Encoder algorithm code is in a directory **mpeg4enc** placed in DM355Codecs/release.

Figure 2-1 shows the sub-directories structure of **mpeg4enc** directory.

Figure 2-1. Component Directory Structure

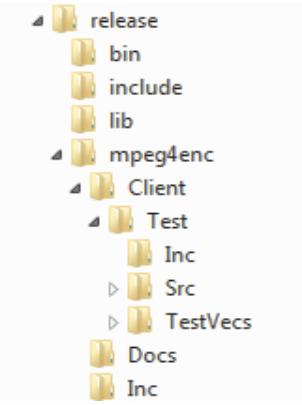


Table 2-1 provides a description of the sub-directories created in the release/mpeg4enc directory.

Table 2-1. Component Directories

Sub-Directory	Description
mpeg4enc /Docs	Contains user guide, datasheet, and release notes
mpeg4enc /Client/Test/Src	Contains application C files
mpeg4enc /Client/Test/Inc	Contains header files needed for the application code
mpeg4enc /Client/Test/TestVecs	Contains test vectors, configuration files
/Include	Contains the include files needed by application and codec.
/lib	Contains MPEG Encoder and other support libraries
/bin	Contains MPEG Encoder executable “mp4enc”

The DM355 MPEG4 encoder library is put into the /release/lib directory and the XDM headers are put in the /release/include directory.

2.3 Building and Running the Sample Test Application on Linux

The sample test application that accompanies this codec component will take YUV input files and dumps encoded output files as specified in the configuration file. To build and run the sample test application, follow these steps:

Step 1. Verify that libmp4venc.a library is present in DM355Codecs/release/lib directory.

Step 2. Verify that support libraries (libcosl.a, libimcopcsl.a,libdm355mm.a, libcmem.a) are present in DM355Codecs/release/lib directory.

Step 3. Change directory to DM355Codecs/release/mpeg4enc/Client/Test/Src and type “make clean” followed by a “make” command. This will use the makefile in that directory to build the test executable mp4enc into the DM355Codecs/release/bin directory.

Note: ARM tool chain, i.e., arm_v5t_le-gcc (ARM gcc) compiler path needs to be set in user's environment path before building the MPEG4 encoder executable.

To run mp4enc executable on DM355 EVM board, use the following instructions.

1. Set up the DM355 EVM Board.

   For information about setting up the DM355 environment, see the *DM355 Getting Started Guide* released in the “doc” directory in the DVSDK release package.

2. Run the MPEG4 Encoder Executable

   - For running the MPEG4 Encoder executable, copy the executable "mp4enc" along with the entire "TestVecs" directory provided with the release package at project/mpeg4enc/Client/Test to the target directory.
   - Copy the kernel modules "dm350mmap.ko" and "cmemk.ko" to the target directory. These modules are provided with the release package in project/bin directory.
   - Copy "loadmodules.sh" provided with release package at project/bin to the target directory.
   - Execute the following commands in this sequence to run the MPEG4 Encoder executable.

     $./loadmodules.sh

     $./mp4enc

     This will run the MPEG4 Encoder with base parameters.
   - To run the MPEG4 Encoder with extended parameters, change the config file in Testvecs.cfg to Testparams.cfg (TestVecs/Config/) and execute:

     $./mp4enc –ext

2.4 Configuration Files

This codec is shipped with:

- A generic configuration file (Testvecs.cfg) – specifies input and output files for the sample test application.
- An Encoder configuration file (Testparams.cfg) – specifies the configuration parameters used by the test application to configure the encoder.

2.4.1 Generic Configuration File

The sample test application shipped along with the codec uses the configuration file, Testvecs.cfg, for determining the input and output files for running the codec. The Testvecs.cfg file is available in the /Client/Test/TestVecs/Config/profile sub-directory.

The format of the Testvecs.cfg file is:

```
X
Config
Input
Output/Reference
```

where:

- X must be set as:
  - 0 - For output dumping
- Config is the Encoder configuration file. For details, see Section 2.4.2.
- Input is the input file name (use complete path).
- Output is the output file name.

A sample Testvecs.cfg file is as follows:

```
0
./TestVecs/Config/profile/Testparams_akiyo_160x128_422.cfg
./TestVecs/Input/akiyo_160x128_422.yuv
./TestVecs/Output/akiyo_160x128_422.bits
```

2.4.2 Encoder Configuration File

The encoder configuration file, Testparams.cfg, contains the configuration parameters required for the encoder. A sample Testparams.cfg file is available in the /Client/Test/TestVecs/Config/profile sub-directory.

A sample Testparams.cfg file is as follows:

```
# New Input File Format is as follows
# <ParameterName> = <ParameterValue> # Comment
#
###############################################################################################
# Parameters
###############################################################################################
ImageHeight      = 128     # Image height in Pels, must be multiple of 16

ImageWidth       = 160     # Image width in Pels, must be multiple of 16

FrameRate        = 30000   # Frame Rate per second*1000 (1-100)

Bitrate          = 737280  # Bitrate(bps)
                             if ZERO then RateControl is OFF.
                             This parameter is ignored for IVIDEO_NONE

ChromaFormat     = 4       # 1 => XDM_YUV_420P, 4 => XDM_YUV_422ILE,
                             Others not supported

subWindowHeight  = 128     # Height of the Subwindow, must be multiple of
                             16

subWindowWidth   = 160     # Width of the Subwindow, must be multiple of
                             16

IntraPeriod      = 30      # Period of I-Frames (N-1) P frames
```

```
                                (Non-negative)

intraDcVlcThr     = 0       # Intra VLC Threshold (0 to 7)

meRange           = 7       # Motion Estimation Range
                              7: ME7
                             31: ME31.
                             Others not supported.

intraThres        = 200     # Threshold for introducing Intra MBs.
                             (0 to 0xFFFF)
                              Use 192 for better quality.

intraAlgo         = 1       # INTRA/INTER Decision Algorithm.
                             0: DM355_MPEG4E_INTRA_INTER_DECISION_LQ_HP
                             1: DM355_MPEG4E_INTRA_INTER_DECISION_HQ_LP

numMBRows         = 5       # Number of MB rows in a Packet.
                             Maximum value is "subWindowHeight/16"

initQ             = 0       # Initial Q (at picture head).
                             0:   automatically determined,
                             2-31:force initial Q

rcQ_MAX           = 31      # Q MAX value  (2-31)

rcQ_MIN           = 2       # Q MIN value  (2-31)
rateFix           = 0       # Q FIX PIC:change Q parameter
                             0: at MB
                             1: at picture start

rateFixRange      = 4       # Q FIX RANGE: Q scale differential range
                             in Q fix mode. If rateFix = 0 then this
                             parameter is ignored.
                             (0-31)

rotation          = 0       # Rotation (anticlockwise)
                             0: No Rotation,
                             1: 90 degree,
                             2: 180 degree,
                             3: 270 degree

meAlgo            = 1       # Motion estimation algorithm
                             0: DM355_MPEG4E_ME_MQ_MP
                             1: DM355_MPEG4E_ME_HQ_MP
                             2: DM355_MPEG4E_ME_HQ_LP
                             3: DM355_MPEG4E_ME_LQ_HP

SkipMBAlgo        = 0       # 0: DM355_MPEG4E_SKIP_MB_LQ_HP
                             1: DM355_MPEG4E_SKIP_MB_HQ_LP

UMV               = 0       # 0: DM355_MPEG4E_UMV_LQ_HP
                             1: DM355_MPEG4E_UMV_HQ_LP

RCAlgo            = 0       # 1: IVIDEO_LOW_DELAY,
                             2: IVIDEO_STORAGE,
                             4: IVIDEO_NONE,
                             6: IMP4VENC_VBR_RATEFIX

IIDC              = 0       # 0: DM355_MPEG4E_INTRA_INTER_BLK_SIZE_LQ_HP
                             1: DM355_MPEG4E_INTRA_INTER_BLK_SIZE_HQ_LP

InitQ_P           = 1       # Initial Q for P frame at the beginning of
                             the frame
                                0: automatically determined,
                             2-31: force initial Q

VBV_size          = 100     # Video rate Buffer verifier size in 16 kb.
                             Number depending on the resolution of video
                             frame, typically equal to (No. of MB's per
                             frame x110) /16000
```

Any field in the IVIDENC1_Params structure (see Section 4.1.1) can be set in the Testparams.cfg file using the syntax shown previously. If you specify additional fields in the Testparams.cfg file, the test application must be appropriately modified to handle these fields.

- If initQ and InitQ_P are not specified through an extended parameter, the default value is taken as 3 and 4, respectively. So if RCAlgo is VBR or VBR with rate fix control, the Qp value is 3 and 4 for I-frame and P-frame respectively.
- VBV_size should be of the order of (number of MB's in a frame X 110) / 16000 for better quality.
- If SkipMBAlgo is enabled, the Intra_thres value should be 192 for the better quality.
- If IntraAlgo is 'DM355_MPEG4E_INTRA_INTER_DECISION_LQ_HP', and if SkipMBAlgo is DM355_MPEG4E_SKIP_MB_HQ_LP, then codec will return an error.
- If MVDataEnable = 1 (extended dynamic parameters) and If IntraAlgo is 'DM355_MPEG4E_INTRA_INTER_DECISION_LQ_HP'', then codec will return an error.
- The best quality is achieved with the following parameter settings

```
{
intraAlgo    =    DM355_MPEG4E_INTRA_INTER_DECISION_HQ_LP
MeAlgo       =    DM355_MPEG4E_ME_HQ_LP
IIDC         =    DM355_MPEG4E_INTRA_INTER_BLK_SIZE_HQ_LP
SkipMBAlg0   =    DM355_MPEG4E_SKIP_MB_HQ_LP
UMV          =    DM355_MPEG4E_UMV_HQ_LP

}
```

- The best performance is achieved with the following parameter settings

```
{
intraAlgo    =    DM355_MPEG4E_INTRA_INTER_DECISION_LQ_HP
MeAlgo       =    DM355_MPEG4E_ME_LQ_HP
IIDC         =    DM355_MPEG4E_INTRA_INTER_BLK_SIZE_LQ_HP
SkipMBAlg0   =    DM355_MPEG4E_SKIP_MB_LQ_HP
UMV          =    DM355_MPEG4E_UMV_LQ_HP

}
```

- Typical bitrates for different resolutions are as follows:

Resolution	WidthxHeight	Typical bitrate(Kbps)
QCIF	176X144	128
QVGA	320X240	256
CIF	352X288	512
VGA	640X480	2000
D1	720X480	3000
4CIF	704X576	4000
720p	1280X720	8000
SXVGA	1280X960	10000

Sample Usage

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

Topic **Page**

3.1 MPEG4 Encoder Client interfacing constraints

The following constraints should be taken into account while implementing the client for the MPEG4 encoder library in this release:

- DMA requirements of MPEG4 Encoder: Current implementation of the MPEG4 encoder uses the following TCCs for its DMA resource requirements along with its associated PaRamSets:

Channel Number	Associated PaRamSet Numbers
12,13,33,34,35,36,37,38,40-50,52-57, 63	12,13,33,34,35,36, 40-50, 52-57, 63 (PaRamSet number = channel number)

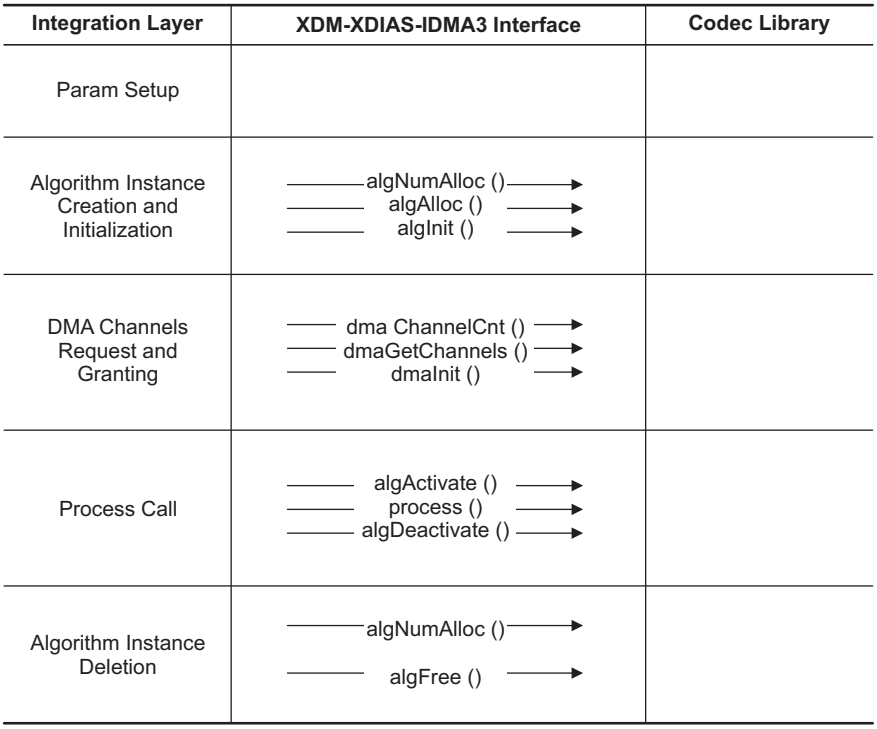
- Apart from the associated PaRamSets, the encoder requires additional 32 PaRamSets, which it gets through the IDMA3 interface from client application.
- Client application must map all the DMA channels used by MPEG4 encoder to the same queue. This is required for the codec to function normally. The codec does not map channels to queue.
- If there are multiple instances of a codec and/or different codec combinations, the application can use the same group of channels and PaRAM entries across multiple codecs. The AlgActivate and AlgDeactivate calls made by client application and implemented by the codecs, perform context save/restore to allow multiple instances of same codec and/or different codec combinations.
- As all codecs use the same hardware resources, only one process call per codec should be invoked at a time (frame level reentrancy). The process call needs to be wrapped within activate and deactivate calls for context switch. Refer to XDM specification on activate/deactivate.
- If multiple codecs are running with frame level reentrancy, the client application has to perform time multiplexing of process calls of different codecs to meet the desired timing requirements between video/image frames.
- The ARM and DDR clock must be set to the required rate for running single or multiple codecs.
- The codec combinations feasibility is limited by processing time (computational hardware cycles) and DDR bandwidth.
- Codec atomicity is supported at frame level processing only. The process call has to run until completion before another process call can be invoked.

3.2 Overview of the Test Application

The test application exercises the IMP4VENC_Params extended class of the MPEG4 Encoder library. The main test application files are mpeg4EncTest355.c and TestAppEncoder.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

3.2.1 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, etc. The test application obtains the required parameters from the encoder configuration files.

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

1. Opens the generic configuration file, Testvecs.cfg, and reads the compliance checking parameter, Encoder configuration file name (Testparams.cfg), input file name, and output/reference file name.
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 IMP4VENC_Params structure based on the values it reads from the Testparams.cfg file
4. Reads the input bit stream into the application input buffer.

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

3.2.2 Algorithm Instance Creation and Initialization

In this logical block, ALG_create() is called by the test application and accepts the various initialization parameters and returns an algorithm instance pointer. The following APIs implemented by the codec are called in sequence by ALG_create():

1. `algNumAlloc()` - To query the algorithm about the number of memory records it requires.
2. `algAlloc()` - To query the algorithm about the memory requirement to be filled in the memory records.

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

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

   In addition, `ALG_create()` uses some APIs that manage memory allocation, such as: `_ALG_allocMemory(),` and `_ALG_freeMemory()`. They are provided in the file `alg_malloc.c`.

   Apart from algorithm memory allocation, the application needs to call the IDMA3_Create() function. This function uses the algorithm instance created in the previous call of ALG_create, and provides the algorithm with the requisite DMA resources. The following APIs implemented by the algorithm are called in the following sequence:
4. `dmaGetChannelCnt()` - To query the algorithm about the number of memory records it requires. In the present implementation, it always defaults to 1.
5. `dmaGetChannels()` - To query the algorithm about the number of additional PaRamSets it requires in the channel records. In the current implementation, the algorithm uses hard-coded channels and its associated TCCs and PaRamSets internally. The client application using the algorithm's IDMA3 interface allocates additional PaRamSets requirements.
6. `dmaInit()` - To initialize the algorithm with continuous PaRamSet addresses allocated to the algorithm during this instance. A sample implementation of this function is included in the idma3_create.c file

3.2.3 Process Call in Single Instance Scenario

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

1. Calls `algActivate()`, which initializes the encoder state and some hardware memories and registers.
2. Sets the input and output buffer descriptors required for the `process()` function call.
3. Calls the `process()` function to encode a single frame of data. The inputs to the process function are input and output buffer descriptors, the pointer to the IVIDENC1_InArgs and `IVIDENC1_OutArgs` structures. `process()` function should be called multiple times to encode multiple frames.
4. Call `algDeactivate()`, which performs releasing of hardware resources and saving of encoder instance values.
5. `process()` is made a blocking call, but an internal OS specific layer enables the process to be pending on a semaphore while hardware performs a complete MPEG4 encode.
6. Other specific details of the `process()` function remain the same as described in section 3.1.3 and the constraints described in section 3.1.1 apply

Note: algActivate () is a mandatory call before first process()call, as it does hardware initialization.

3.2.4 Algorithm Instance Deletion

Once encoding is complete, the test application must delete the current algorithm instance. The following APIs are called in sequence:

1. `algNumAlloc()` - To query the algorithm about the number of memory records it used
2. `algFree()` - To query the algorithm for the memory record information and then free them up for the application

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.

3.3 Usage in Multiple Instance Scenario

For client applications that support multiple instances of MPEG4 encoder, initialization and process calls are altered. One of the main issues in converting a single instance encoder to a multiple instance encoder is resource arbitration and data integrity of shared resources between various codec instances. Resources that are shared between instances and need to be protected include:

- DMA channels and PaRamSets
- MPEG4 Hardware Co-Processors and their memory areas

To protect one instance of the MPEG4 encoder from overwriting into these shared resources when the other instance is actually using it, the application needs to implement mutex in the test applications. You can implement custom resource sharing mutex and call algorithm APIs after acquiring the corresponding mutex. Since all codecs (JPEG encoder/decoder and MPEG-4 encoder/decoder) use the same hardware resources, only one codec instance can run at a time.

Here are some of the API combinations that need to be protected with single mutex.

1. `dmaInit()` of one instance initializes DMA resources when the other instance is actually active in its `process()` function.
2. `control()` call of one instance sets post-processing function properties by setting the command length, etc., when the other instance is active or has already set its post processing properties.
3. `process()` call of one instance tries to use the same hardware resources [co-processor and DMA] when the other instance is active in its process() call.

If multiple instances of the MPEG4 encoder are used in parallel, the hardware must be reset between every process call and algorithm memory to be restored. This is achieved by calling `algActivate()` and `algDeactivate()` before and after process() calls.

Thus, the Process call section as explained previously changes to include both `algActivate()` and `algDeactivate()` as mandatory calls of the algorithm.

3.3.1 Process Call With algActivate and algDeactivate

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

1. Sets the input and output buffer descriptors required for the `process()` function call.
2. Calls `algActivate()`, which initializes the encoder state and some hardware memories and registers.
3. Calls the `process()` function to encode a single frame of data. The inputs to the process function are input and output buffer descriptors, and the pointer to the `IVIDENC1_InArgs` and `IVIDENC1_OutArgs` structures.
4. Calls `algDeactivate()`, which releases hardware resources and saves the decoder instance values.
5. Other specific details of the `process()` function remain the same as described in section 3.1.3 and the constraints described in section 3.1.1 are applicable.

Note: In a multiple instance scenario, `algActivate()` and `algDeactivate()` are mandatory function calls before and after `process()`, respectively.

3.4 Usage for Motion Vector Access

For client applications that support motion vector access, the initialization and process calls are same as explained in Section 3.2.

3.4.1 Description

The Motion Vector Access API is part of the XDM `process()` call that the application uses to encode a frame. A run time parameter MVDataEnable is provided as a part of dynamic parameters, which can be set or reset at a frame level at run time. Setting this flag to 1 indicates that the motion vectors access is needed. When this parameter is set to 1, the `process()` call returns the motion vector data in the buffer provided by the application.

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

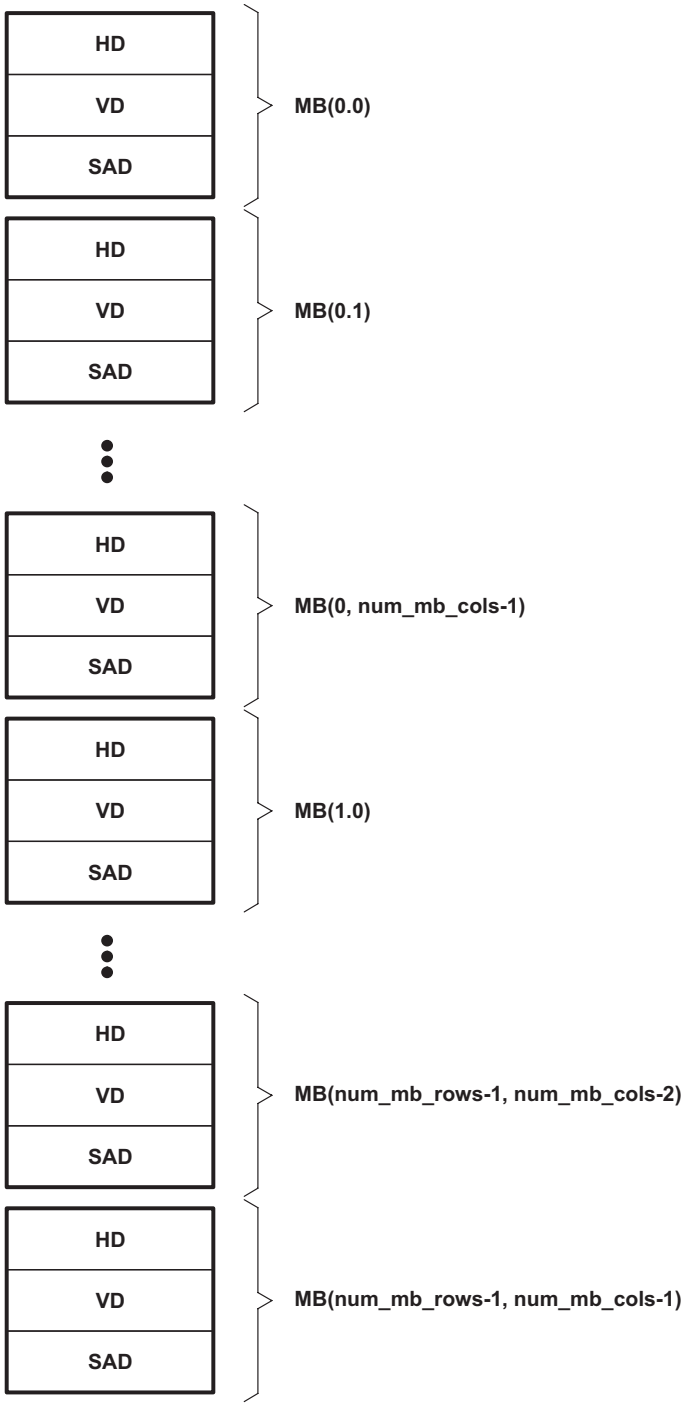
Motion Vector Horizontal Displacement (HD)	Signed 16 - bit integer
Motion vector vertical displacement (VD)	Signed 16 - bit integer
SAD	Unsigned 32 - bit integer

Notes:

- Motion vector access is not supported with intraAlgo = DM355_MPEG4E_INTRA_INTER_DECISION_LQ_HP
- The current version of the MPEG4 encoder stores the SAD (sum of Absolute differences) in place of SSE

The API returns the motion vector data in a single buffer with these three values interleaved in contiguous memory as shown in the following Figure 3-2.

Figure 3-2. : Motion Vector and SAD Buffer Organization



3.4.2 Usage

The following sequence should be followed for motion vector access:

1. In the dynamic parameters, set the flag to access MV data:

```
/* This structure defines the run time parameters for MP4VEnc object */
MP4VENC_DynamicParams   ext_dynamicParams;

/* Enable MV access */
```

```
ext_dynamicParams ->MVDataEnable = 1;

/* Control call to set the dynamic parameters */
control(.., XDM_SETPARAMS,..)
```

2. Allocate output buffers and define the output buffer descriptors:

```
/* Output Buffer Descriptor variables */
XDM_BufDesc  outputBufDesc;

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

If MV access is enabled in step 1, this call will return the buffer info as minNumOutBufs=2, along with the minimal buffer sizes.

```
/* Initialize the output buffer descriptor */
outputBufDesc.numBufs = status.videncStatus.bufInfo.minNumOutBufs;
/* Stream Buffer */
outputBufDesc.bufs[0]     =  streamDataPtr; //pointer to mpeg4 bit stream
outputBufDesc.bufSizes[0] = status.videncStatus.bufInfo.minOutBufSize[0];
/* MV Buffer */
outputBufDesc.bufs[1]     =  mvDtataPtr; //pointer to MV data
outputBufDesc.bufSizes[1] = status.videncStatus.bufInfo.minOutBufSize[1];
```

3. Call the frame encode API:

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

After this call, the buffer outputBufDesc.bufs[1] will have the Motion vector data. This API will return the size of the MV array in outArgs.mvDataSize.

As shown in Figure 3-2, the API uses a single buffer to store the motion vector data. The buffer will have the three values (HD, VD, SAD) interleaved in contiguous memory.

Define a structure:

```
struct motion_mbdata
{
       short MVx;
       short MVy;
       unsigned int SAD;
} ;

motion_mbdata  *mbMV_data =  outputBufDesc.bufs[1];

num_mb_rows = frameRows / 16;
num_mb_cols  = frameCols / 16;

for (i = 0; i < num_mb_rows; i++)
{
      for (j = 0; j < num_mb_cols; j++)
     {
 HD for mb(i, j) = mbMV_data ->MVx;

VD for mb(i, j) = mbMV_data ->MVy;

SAD for mb(i,j) = mbMV_data ->SSE;

mbMV_data ++;
      }
}
```

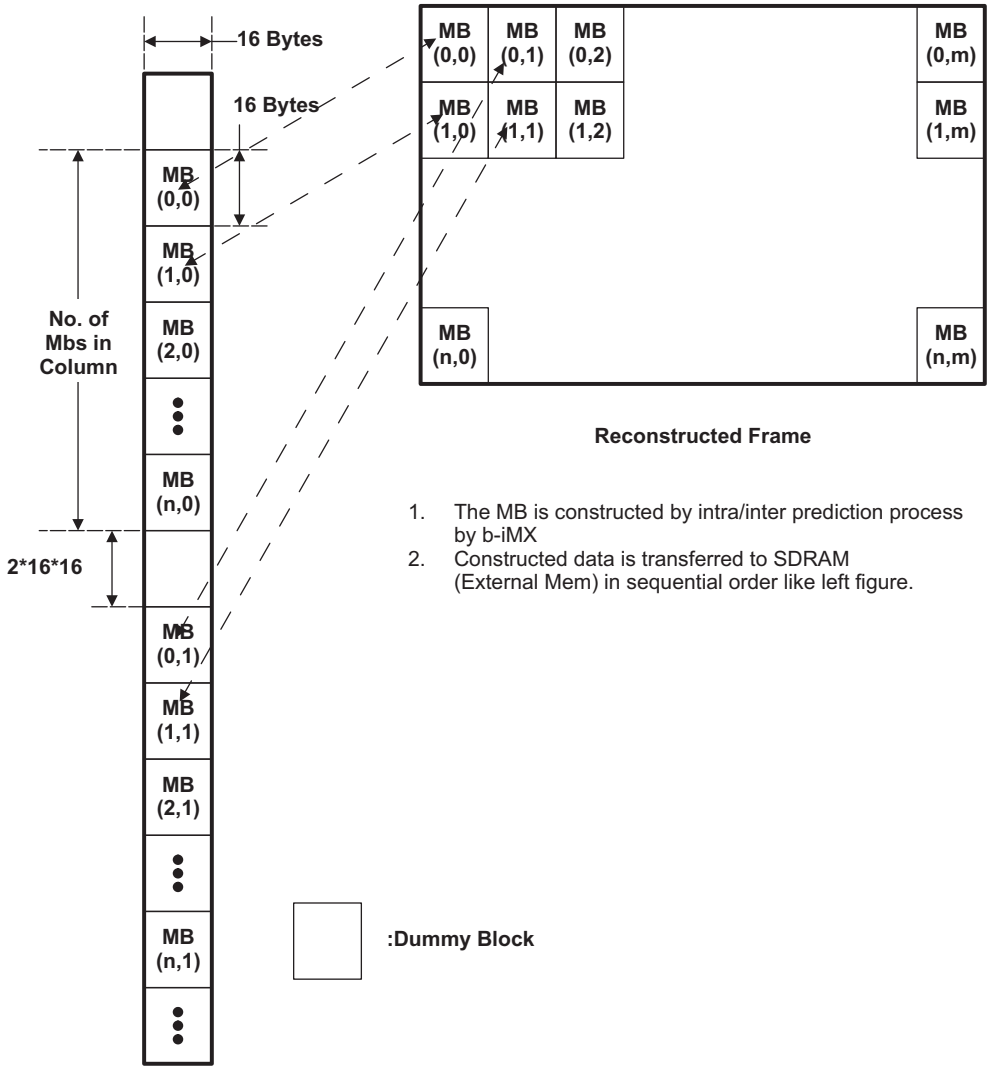
Notes:

- The motion vectors are with fullpel (integer pel) resolution.
- SSE = (Ref(i,j) – Src(i,j)) 2, where Ref is the macro block of the reference region and Src is the macro block of the source image.
- Current version of the MPEG4 encoder stores the SAD (sum of Absolute differences) in place of SSE.
- The motion vectors seen in the encoded stream are based on the best coding decision, which is a combination of the motion estimation and mode decisions. The MV buffer returns the results of the motion estimation in fullpel resolution (lowest SAD), which may be different from the motion vectors seen in the bit stream
  - Some macro blocks in a P-frame may be coded as Intra macro blocks based on the post motion estimation decisions. In this case, the motion vectors computed in the motion estimation stage (assuming that this macro block is inter) will be returned.
  - Some macro blocks in a P-frame may be "Not Coded" (i.e., skipped). In this case, motion vectors of (0,0) and SAD corresponding to (0,0) motion vector are returned.
  - For I-frames, motion vectors are not returned and outArgs.mvDataSize = 0.

3.5 Accessing Reconstruction Buffer Data

The structure of reconstruction buffer used in the MPEG4 encoder is shown in the Figure 3-3. The reconstructed data is not really stored in YUV 420P format. Luma data is stored continuously in "outArgs.reconBufs.bufDesc[0].buf" buffer, while chroma data is stored in "outArgs.reconBufs.bufDesc[1].buf" as interleaved Cb and Cr format.

Figure 3-3. Reconstruction Buffer



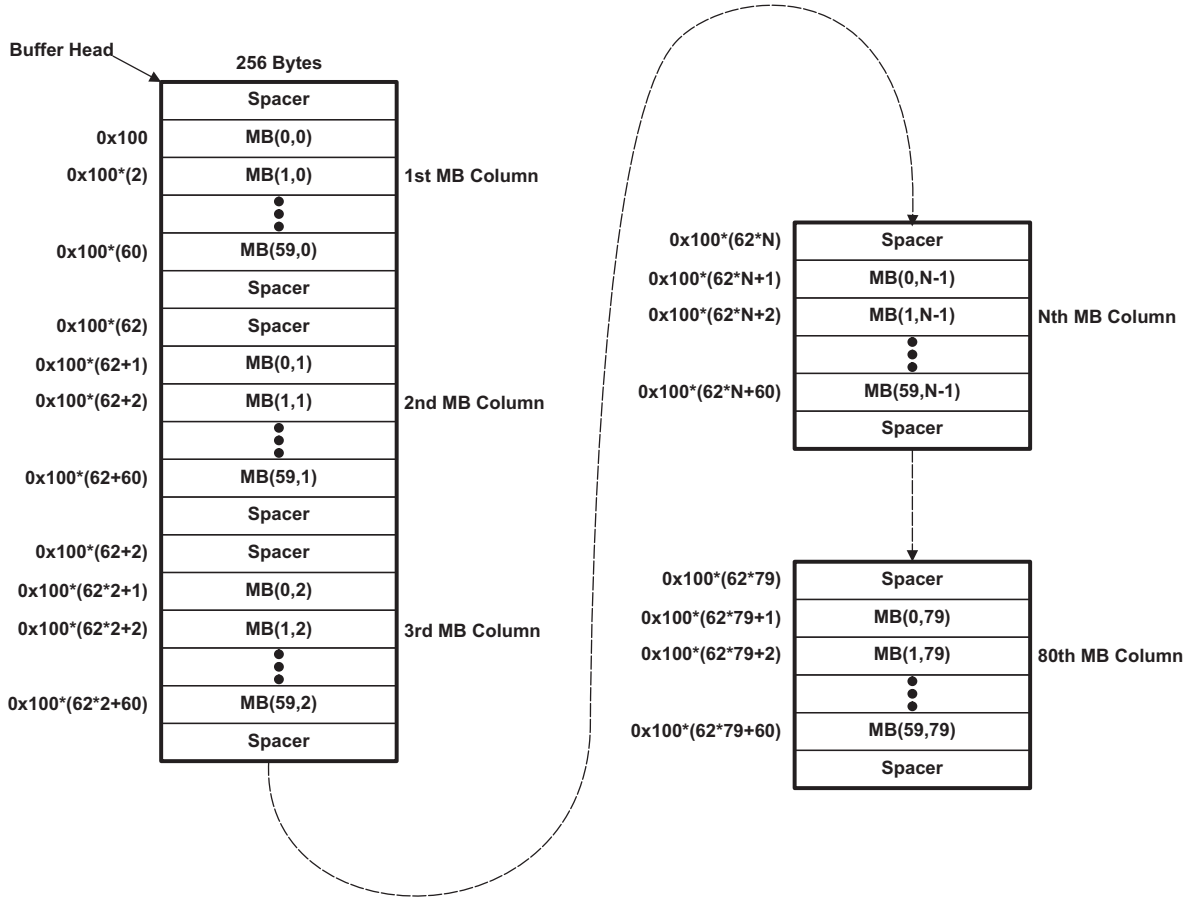
The MBs stored in the reconstruction buffer are actually column MBs in display frame. Also the MB data are stored continuously in reconstruction buffer (ie 256 bytes/MB of luma of entire frame and then follows 128bytes/MB of chroma data).

So to access first MB of the frame, the offset will be 16*16 for luma and 16*8 for chroma. Similarly to access the second MB (in actual display frame), offset of ((No of MBs in column + 2) * 256 + 256) is added to the base reconstruction buffer pointer for luma.

Figure 3-4 and Figure 3-5 illustrate the format of the recon buffer taking SXVGA as an example.

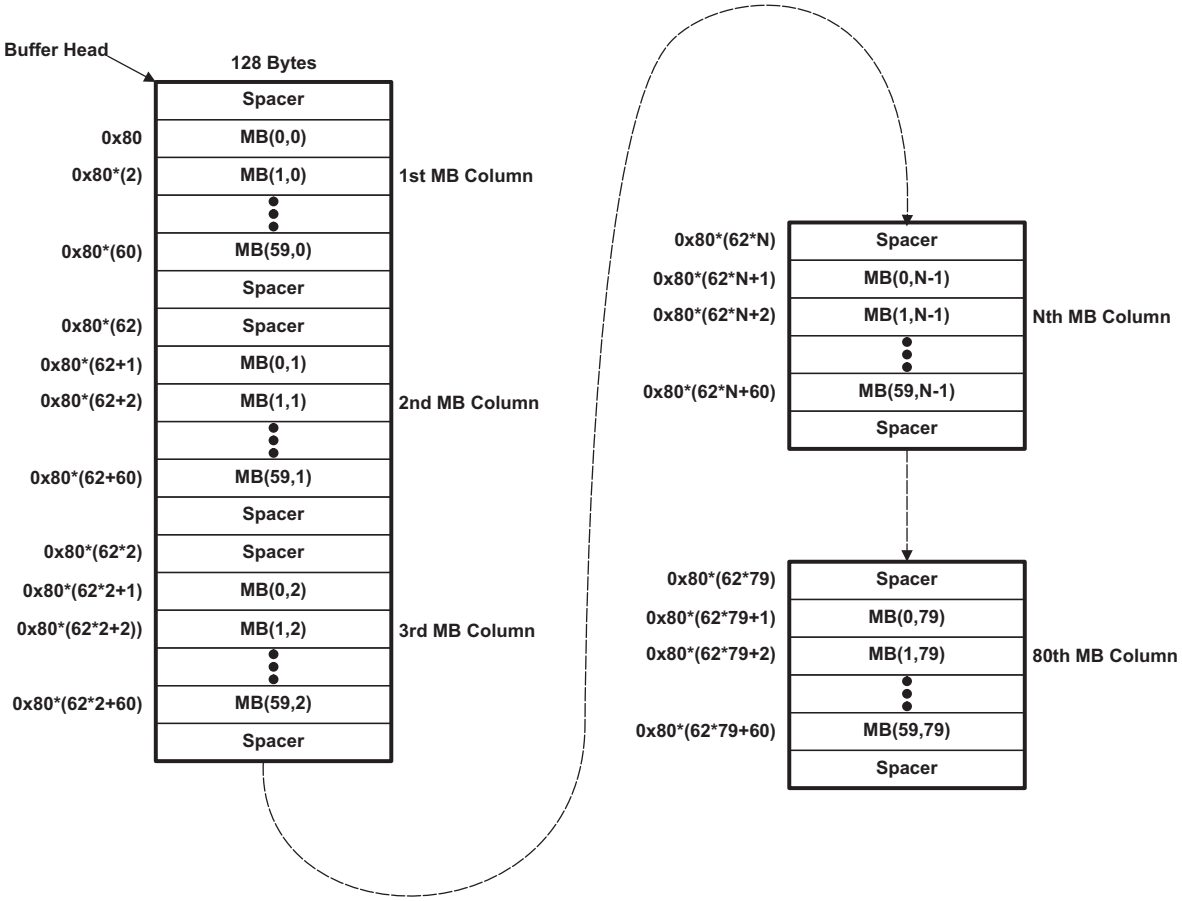
For Luma

Figure 3-4. Reconstruction Buffer for Luma



For Chroma

Figure 3-5. Reconstruction Buffer for Chroma



Below is a sample application code to extract luma and chroma data from the recon buffer and dump into output file in the planar YUV420 format.

```
mbSizeY = extn_params.subWindowHeight >> 4;
mbSizeX = extn_params.subWindowWidth >> 4;
dummy_ptr_lum = (unsigned char *)(outArgs.reconBufs.bufDesc[0].buf + 16*16);
dummy_ptr_chr = (unsigned char *)(outArgs.reconBufs.bufDesc[1].buf + 16*8);
/*
outArgs.reconBufs.bufDesc[0].buf   -> Base address for the luma recon buffer
outArgs.reconBufs.bufDesc[1].buf   -> Base address for the chroma recon buffer
temp_buffer_bk is the base pointer address for the output buffer for YUV420 planar
allocated with the memory of (width * height * 1.5)
*/
lumaOffset = 16*16*(mbSizeY+2);
chrOffset = lumaOffset /2;
for(i=0;i<(extn_params.subWindowWidth>>4) ;i++)
{
     for(j=0;j<mbSizeY;j++)
     {
       temp_buffer= temp_buffer_bk + (extn_params.subWindowWidth*16*j) +  (i*16);
       temp_lum = dummy_ptr_lum + ( lumaOffset *i) + (256*j);
       temp_buffer_cb = temp_buffer_cb_bk+(extn_params.subWindowWidth*4*j) + (i*8);
       temp_buffer_cr = temp_buffer_cr_bk+(extn_params.subWindowWidth*4*j) + (i*8);
       temp_chr =(dummy_ptr_chr + (chrOffset *i)) + (128*j);

/* Extract Luma*/
       for(k=0;k<16;k++)
       {
             for(l=0;l<16;l++)
             {
                 temp_buffer[k*extn_params.subWindowWidth+l] =
                 temp_lum[k*16+l];
            }
```

```
            }

             /* Extract Chroma*/
            for(k=0;k<8;k++)
            {
                  for(l=0;l<8;l++)
                  {
                      temp_buffer_cb[k*(extn_params.subWindowWidth >> 1) + l] =
                      temp_chr[k*16+2*l];
                      temp_buffer_cr[k*(extn_params.subWindowWidth >> 1) + l] =
                      temp_chr[k*16+(2*l)+1];
                  }
             }
        }
    }

    fwrite(temp_buffer_bk, 1, ((extn_params.subWindowWidth * extn_params.subWindowHeight *3)>>1),
    fReconBuffer);
    ..............................................
```

API Reference

This chapter describes the Application Programming Interfaces (APIs).

Topic **Page**

4.1 Symbolic Constants and Enumerated Data Types

This section summarizes all the symbolic constants specified as either #define macros and/or enumerated C data types. Described alongside the macro or enumeration is the semantics or interpretation of the same in terms of what value it stands for and what it means.

Table 4-1. List of Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
IVIDEO_FrameType	IVIDEO_I_FRAME	0	Intra coded frame
	IVIDEO_P_FRAME	1	Forward inter coded frame
	IVIDEO_B_FRAME	2	Bi-directional inter coded frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_IDR_FRAME	3	Intra coded frame that can be used for refreshing video content. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_II_FRAME	4	Interlaced Frame, both fields are I frames. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_IP_FRAME	5	Interlaced Frame, first field is an I frame, second field is a P frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_IB_FRAME	6	Interlaced Frame, first field is an I frame, second field is a B frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_PI_FRAME	7	Interlaced Frame, first field is a P frame, second field is a I frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_PP_FRAME	8	Interlaced Frame, both fields are P frames. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_PB_FRAME	9	Interlaced Frame, first field is a P frame, second field is a B frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_BI_FRAME	10	Interlaced Frame, first field is a B frame, second field is an I frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_BP_FRAME	11	Interlaced Frame, first field is a B frame, second field is a P frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_BB_FRAME	12	Interlaced Frame, both fields are B frames. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_MBAFF_I_FRAME	13	Intra coded MBAFF frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_MBAFF_P_FRAME	14	Forward inter coded MBAFF frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_MBAFF_B_FRAME	15	Bidirectional inter coded MBAFF frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_MBAFF_IDR_FRAME	16	Intra coded MBAFF frame that can be used for refreshing video content. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_FRAMETYPE_DEFAULT	0	Default set to IVIDEO_I_FRAME
IVIDEO_ContentType	IVIDEO_CONTENTTYPE_NA	-1	Content type is not applicable. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_PROGRESSIVE	0	Progressive video content. Default value.
	IVIDEO_INTERLACED	1	Interlaced video content. Not supported in this version of the MPEG4 Encoder.

Table 4-1. List of Enumerated Data Types (continued)

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
IVIDEO_RateControlPreset	IVIDEO_LOW_DELAY	1	Constant Bit Rate (CBR) control for video conferencing. (i.e., CBR with frames dropped based on VBV buffer occupancy.)
	IVIDEO_STORAGE	2	Variable Bit Rate (VBR) control for local storage (DVD) recording. Default value. (i.e., CVBR: similar to IVIDEO_LOW_DELAY but no frame skips)
	IVIDEO_TWOPASS	3	Two pass rate control for non real time applications. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_NONE	4	No rate control algorithm.(i.e., VBR mode: Fixed Qp values)
	IVIDEO_USER_DEFINED	5	User defined through extended parameters. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_RATECONTROLPRESET_DEFAULT	1	Set to IVIDEO_LOW_DELAY
IVIDEO_SkipMode	IVIDEO_FRAME_ENCODED	0	Input content encoded
	IVIDEO_FRAME_SKIPPED	1	Input content skipped, that is, not encoded
	IVIDEO_SKIPMODE_DEFAULT	0	Default value set to IVIDEO_FRAME_ENCODE
XDM_DataFormat	XDM_BYTE	1	Big endian stream
	XDM_LE_16	2	16-bit little endian stream. Not supported in this version of the MPEG4 Encoder.
	XDM_LE_32	3	32-bit little endian stream. Not supported in this version of the MPEG4 Encoder.
XDM_ChromaFormat	XDM_CHROMA_NA	-1	Chroma format not applicable. Not supported in this version of the MPEG4 Encoder.
	XDM_YUV_420P	1	YUV 4:2:0 planar
	XDM_YUV_422P	2	YUV 4:2:2 planar. Not supported in this version of the MPEG4 Encoder.
	XDM_YUV_422IBE	3	YUV 4:2:2 interleaved (big endian). Not supported in this version of the MPEG4 Encoder.
	XDM_YUV_422ILE	4	YUV 4:2:2 interleaved (little endian)
	XDM_YUV_444P	5	YUV 4:4:4 planar. Not supported in this version of the MPEG4 Encoder.
	XDM_YUV_411P	6	YUV 4:1:1 planar. Not supported in this version of the MPEG4 Encoder.
	XDM_GRAY	7	Gray format. Not supported in this version of the MPEG4 Encoder.
	XDM_RGB	8	RGB color format. Not supported in this version of the MPEG4 Encoder.
XDM_CmdId	XDM_GETSTATUS	0	Query algorithm instance to fill Status structure
	XDM_SETPARAMS	1	Set run time dynamic parameters via the DynamicParams structure
	XDM_RESET	2	Reset the algorithm.
	XDM_SETDEFAULT	3	Initialize all fields in Params structure to default values specified in the library
	XDM_FLUSH	4	Handle end of stream conditions. This command forces algorithm instance to output data without additional input.
	XDM_GETBUFINFO	5	Query algorithm instance regarding the properties of input and output buffers
	XDM_GETVERSION	6	Query the algorithm's version. Not supported in this version of the MPEG4 Encoder.

Table 4-1. List of Enumerated Data Types (continued)

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
XDM_EncodingPreset	XDM_DEFAULT	0	Default setting of the algorithm specific creation time parameters. This uses XDM_HIGH_QUALITY settings.
	XDM_HIGH_QUALITY	1	Set algorithm specific creation time parameters for high quality (default setting).
	XDM_HIGH_SPEED	2	Set algorithm specific creation time parameters for high speed.
	XDM_USER_DEFINED	3	User defined configuration using advanced parameters. This uses XDM_HIGH_QUALITY settings in case of non-extended params.
XDM_EncMode	XDM_ENCODE_AU	0	Encode entire access unit, including the headers. Default value.
	XDM_GENERATE_HEADER	1	Encode only header.
XDM_ErrorBit	XDM_APPLIEDCONCEALMENT	9	Bit 9 1 : Applied concealment 0 : Ignore
	XDM_INSUFFICIENTDATA	10	Bit 10 1 : Insufficient data 0 : Ignore
	XDM_CORRUPTEDDATA	11	Bit 11 1 : Data problem/corruption 0 : Ignore
	XDM_CORRUPTEDHEADER	12	Bit 12 1 : Header problem/corruption 0 : Ignore
	XDM_UNSUPPORTEDINPUT	13	Bit 13 1 : Unsupported feature/parameter in input 0 : Ignore
	XDM_UNSUPPORTEDPARAM	14	Bit 14 1 : Unsupported input parameter or configuration 0 : Ignore
	XDM_FATALERROR	15	Bit 15 1 : Fatal error (stop encoding) 0 : Recoverable error

Table 4-2. Data Structures

Title **Page**

Table 4-2. Data Structures (continued)

4.1.1 Common XDM Data Structures

This section includes the following common XDM data structures:

- XDM1_BufDesc
- XDM1_SingleBufDesc
- XDM1_AlgBufInfo
- IVIDEO1_BufDescIn
- IVIDENC1_Fxns
- IVIDENC1_Params
- IVIDENC1_DynamicParams
- IVIDENC1_InArgs
- IVIDENC1_Status
- IVIDENC1_OutArgs

XDM1_BufDesc

Description

This structure defines the buffer descriptor for input and output buffers in XDM1.0

Fields

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

XDM1_SingleBufDesc

Description

This structure defines the single buffer descriptor for input and output buffers in XDM1.0

Fields

Field	Datatype	Input/Output	Description
`*buf`	`XDAS_Int8`	Input	Pointer to a buffer address
`bufSize`	`XDAS_Int32`	Input	Size of buf in 8-bit bytes
`accessMask`	`XDAS_Int32`	Input	Mask filled by the algorithm, declaring how the buffer was accessed by the algorithm process

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	Datatype	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

IVIDEO1_BufDescIn

Description

This structure defines the Buffer descriptor for input video buffers.

Fields

Field	Datatype	Input/ Output	Description
numBufs	XDAS_Int32	Input	Number of buffers in bufDesc[ ]
frameWidth	XDAS_Int32	Input	Width of the video frame
frameHeight	XDAS_Int32	Input	Height of the video frame
framePitch	XDAS_Int32	Input	Frame pitch used to store the frame. Not Supported in this version of MPEG4 encoder
bufDesc[XDM_MAX_IO_BUFFERS]	XDM1_SingleBufDesc	Input	Picture buffers

IVIDENC1_Fxns

Description

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

Fields

Field	Datatype	Input/ Output	Description
ialg	IALG_Fxns	Input	Structure containing pointers to all the XDAIS interface functions. For more details, see *TMS320 DSP Algorithm Standard API Reference* (SPRU360).
*process	XDAS_Int32	Input	Pointer to the process() function.
*control	XDAS_Int32	Input	Pointer to the control() function.

IVIDENC1_Params

Description

This structure defines the creation parameters for an algorithm instance object.

Fields

Field	Datatype	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
encodingPreset	XDAS_Int32	Input	Encoding preset. See XDM_EncodingPreset enumeration for details. The default is XDM_HIGH_QUALITY. XDM_HIGH_QUALITY (1) : for *Very High Quality and Low Performance*. When the number of MBs in VOP > 2668 , quality will be reduced internally in the codec to achieve better performance (more fps). XDM_HIGH_SPEED(2): for *Moderate Quality and Moderate Performance*. XDM_USER_DEFINED(3): for using the user-defined extended parameters. Apart from these options, the encoder supports the following enumerations as well. IMP4VENC_HIGH_QUALITY_MODERATE_PERFORMANCE (4): for *High Quality and Moderate Performance*. IMP4VENC_HIGHEST_QUALITY_LOWEST_PERFORMANCE (5): for *Highest Quality and Lowest Performance*. When number of MBs in VOP > 2184, quality will be reduced internally in the codec to achieve better performance (more fps). IMP4VENC_LOW_QUALITY_HIGHEST_PERFORMANCE (6): for *Lowest Quality and Highest Performance*. Note: encodingPreset MUST be set to XDM_USER_DEFINED, if you want to set quality/performance tools through extended parameters.
rateControlPreset	XDAS_Int32	Input	Rate control preset: See IVIDEO_RateControlPreset enumeration for details. IVIDEO_TWOPASS and IVIDEO_USER_DEFINED are not supported in this version of encoder. Apart from this, IMP4VENC_VBR_RATEFIX is used to set rate control to Variable Bit Rate (VBR) with rate fix range control. Relevant extended parameters should be set to use any of these rate controls. The default is IVIDEO_STORAGE.
maxHeight	XDAS_Int32	Input	Height of the Input stream in pixels. The maximum value supported is 960. This should be same as the value of inputHeight for the IVIDENC1_DynamicParams structure.

Field	Datatype	Input/ Output	Description
maxWidth	XDAS_Int32	Input	Width of the Input stream in pixels. The maximum value supported is 1280. The minimum value supported is 160 without UMV enabled. The minimum value supported is 192 with UMV. This should be same as the value of inputWidth for the IVIDENC1_DynamicParams structure.
maxFrameRate	XDAS_Int32	Input	Frame rate in fps * 1000. Maximum frame rate supported in this version is 100*1000 for rateControl preset = IMP4VENC_VBR_RATEFIX.
maxBitRate	XDAS_Int32	Input	Bit rate to be used for encoding in bits per second.
dataEndianness	XDAS_Int32	Input	Endianness of input data. See XDM_DataFormat enumeration for details. Only XDM_BYTE is supported in this version
maxInterFrameInterval	XDAS_Int32	Input	Distance from I-frame to P-frame: Only 0 and 1 values are supported in this version of encoder. The default value is 0.
inputChromaFormat	XDAS_Int32	Input	Input chroma format. See XDM_ChromaFormat enumeration for details. Only XDM_YUV_422ILE and XDM_YUV_420P are supported in this version. The default is XDM_YUV_422ILE.
inputContentType	XDAS_Int32	Input	Input content type. See IVIDEO_ContentType enumeration for details. Only IVIDEO_PROGRESSIVE is supported in this version. The default is IVIDEO_PROGRESSIVE.
reconChromaFormat	XDAS_Int32	Input	Chroma formats for the reconstruction buffers. Only XDM_YUV_420P (Default) is supported in this version of encoder, though the chromaformat is not exactly YUV 4:2:0 P. For more details, see Section 3.5 for more details.

Notes:

- If maxBitRate = 0 and if RateControlPreset is VIDEO_LOW_DELAY, maxBitRate is taken as 4mbps
- If maxBitRate = 0, and RateControlPreset is OTHER THAN VIDEO_LOW_DELAY, rateFix is made 1 (fixed Qp)
- If rateFix = 1 (For fixed Qp), then AC prediction is enabled.

IVIDENC1_DynamicParams

Description This structure defines the run time parameters for an algorithm instance object.

Fields

Field	Datatype	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
inputHeight	XDAS_Int32	Input	Height of input frame in pixels. Not supported in this version of the encoder. This should be same as the value of maxHeight for the IVIDENC1_Params structure.
inputWidth	XDAS_Int32	Input	Width of input frame in pixels. Not supported in this version of the encoder. This should be same as the value of maxWidth for the IVIDENC1_Params structure.
refFrameRate	XDAS_Int32	Input	Reference or input frame rate in fps. Not supported in this version of the encoder.
targetFrameRate	XDAS_Int32	Input	Target frame rate in fps * 1000. For example, if the frame rate is 30, set this field to 30000.
targetBitRate	XDAS_Int32	Input	Target bit rate in bits per second. For example, if the bit rate is 2 Mbps, set this field to 2097152.
intraFrameInterval	XDAS_Int32	Input	Interval between two consecutive intra frames. 0 : No inter frames (all intra frames) n : n-1 frames coded as p-frames between every two I-frames.
generateHeader	XDAS_Int32	Input	Encode entire access unit or only header. See XDM_EncMode enumeration for details. The default is XDM_ENCODE_AU.
captureWidth	XDAS_Int32	Input	If the field is set to: 0 : Encoded image width is used as pitch. Any non-zero value, capture width is used as pitch (if capture width is greater than image width). Not supported in this version of the encoder.
forceFrame	XDAS_Int32	Input	Force the current (immediate) frame to be encoded as a specific frame type. Not supported in this version of the encoder.
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. Not supported in this version of the encoder.

Field	Datatype	Input/ Output	Description
mbDataFlag	XDAS_Int32	Input	Flag to indicate that the algorithm should use MB data supplied in additional buffer within inBufs. Not supported in this version of the encoder.

IVIDENC1_InArgs

Description

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

Fields

Field	Datatype	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 bitstream frames. Remarks: This is useful when frames require buffering (e.g., B frames), and to support buffer management. When there is no re-ordering, IVIDENC1_OutArgs::outputID will be the same as this inputID field. Remarks: Zero (0) is not a supported inputID. This value is reserved for cases when there is no output buffer provided.
topFieldFirstFlag	XDAS_Int32	Input	Flag to indicate the field order in interlaced content. Valid values are XDAS_TRUE and XDAS_FALSE. This is not supported in this version of encoder.

IVIDENC1_Status

Description

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

Fields

Field	Datatype	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
extendedError	XDAS_Int32	Output	Extended error code. See XDM_ErrorBit enumeration for details.
data	XDM1_SingleBufDesc	Input	Buffer descriptor for data passing. Not supported in this version.
bufInfo	XDM_AlgBufInfo	Output	Input and output buffer information. See XDM_AlgBufInfo data structure for details.

IVIDENC1_OutArgs

Description

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

Fields

Field	Datatype	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
extendedError	XDAS_Int32	Output	Extended error code. See XDM_ErrorBit enumeration for details.
bytesGenerated	XDAS_Int32	Output	The number of bytes generated after each process/encode call
encodedFrameType	XDAS_Int32	Output	Frame types for video. See IVIDEO_FrameType enumeration for details. Only IVIDEO_I_FRAME, IVIDEO_P_FRAME are supported in this version.
inputFrameSkip	XDAS_Int32	Output	Frame skipping modes for video. See IVIDEO_SkipMode enumeration for details.
outputID	XDAS_Int32	Output	Identifier to attach with the corresponding encoded bitstream frames.
encodedBuf	XDM1_SingleBufDesc	Output	Not supported in this version.
reconBufs	IVIDEO1_BufDesc	Output	Pointer to reconstruction buffer descriptor. For more details, see Section 3.5.

4.1.2 MPEG4 Encoder Data Structures

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

- IMP4VENC_Params
- IMP4VENC_DynamicParams
- IMP4VENC_InArgs
- IMP4VENC_OutArgs
- DM355_MPEG4E_ERROR

IMP4VENC_Params

Description

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

Fields

Field	Datatype	Input/ Output	Description
videnc_params	IVIDENC1_Params	Input	See IVIDENC1_Params data structure for details
subWindowHeight	XDAS_Int32	Input	Height of the Subwindow, must be multiple of 16. It should be less than maxHeight specified in IVIDENC1_params
subWindowWidth	XDAS_Int32	Input	Width of the Subwindow, must be multiple of 16. Must be less than maxWidth specified in IVIDENC1_params.
intraFrameInterval	XDAS_Int32	Input	Period of I-Frames (N-1) P frames (Non-negative)
intraDcVlcThr	XDAS_Int32	Input	Intra VLC Threshold (0 to 7). Maximum value supported is 7 Minimum value supported is 0. Use 0 for better quality.
rotation	XDAS_Int32	Input	Rotation (anticlockwise): 0: No Rotation 1: 90 degree 2: 180 degree 3: 270 degree Other values are not supported.
intraThres	XDAS_Int32	Input	Threshold for introducing Intra MBs. Maximum value supported is 0xFFFF. Minimum value supported is 0. Use a value of 192 for better quality.
intraAlgo	XDAS_Int32	Input	INTRA/INTER Decision Algorithm. DM355_MPEG4E_INTRA_INTER_DECISION_LQ_HP: for low quality and high performance DM355_MPEG4E_INTRA_INTER_DECISION_HQ_LP: for high quality and low performance Use DM355_MPEG4E_INTRA_INTER_DECISION_HQ_LP better quality.
numMBRows	XDAS_Int32	Input	Number of MB rows in a Packet. Maximum value is subWindowHeight/16. This indicates the packet size. Minimum value supported is 1.
initQ	XDAS_Int32	Input	Initial Q (at picture head). 0:automatically determined, 2-31:force initial Q. This is for I frame quantization. (Default value is 3) This should have a value less than rcQ_MAX and more than rcQ_MIN.

Field	Datatype	Input/ Output	Description
rcQ_MAX	XDAS_Int32	Input	Q MAX value Maximum value supported is 31 Minimum value supported is 2
rcQ_MIN	XDAS_Int32	Input	Q MIN value Maximum value supported is 31 Minimum value supported is 2
rateFix	XDAS_Int32	Input	Q FIX PIC: change Q parameter 0: at MB, 1: at picture start. Other values not supported.
rateFixRange	XDAS_Int32	Input	Q FIX RANGE: Q scale differential range in Q fix mode. If rateFix = 0 then this parameter is ignored Maximum value supported is 31 Minimum value supported is 2 Other values are not supported.
VBV_size	XDAS_Int32	Input	Video buffer verifier size in 16 kb. (This parameter's value is based on the resolution of the video.)
InitQ_P	XDAS_Int32	Input	Initial Q (at picture head). 0: Automatically determined 2-31: Force initial Q This is for P frame quantization. (Default value is 4) **Note:** This should have a value of less than rcQ_MAX and more than rcQ_MIN
meRange	XDAS_Int32	Input	Motion Estimation Range 7: ME7 31: ME31 Others not supported. **Note:** This parameter is not used in the current version, as it is decided by meAlgo parameter.
MeAlgo	XDAS_Int32	Input	Motion estimation algorithm DM355_MPEG4E_ME_LQ_HP: For lowest quality and highest performance. DM355_MPEG4E_ME_MQ_MP: For moderate quality and moderate performance. DM355_MPEG4E_ME_HQ_MP: For high quality and moderate performance. DM355_MPEG4E_ME_HQ_LP: For highest quality and lowest performance.

Field	Datatype	Input/ Output	Description
SkipMBAlgo	XDAS_Int32	Input	P Skip MB algorithm DM355_MPEG4E_SKIP_MB _LQ_HP: non-Bonus Skip MB For low quality and high performance DM355_MPEG4E_SKIP_MB _HQ_LP: Bonus Skip MB For high quality and low performance
UMV	XDAS_Int32	Input	Unrestricted motion vector DM355_MPEG4E_UMV_LQ_ HP: Disable DM355_MPEG4E_UMV_HQ _LP: Enable **Note:** If UMV is enabled, the minimum input stream width supported is 192.
IIDC	XDAS_Int32	Input	DM355_MPEG4E_INTRA_IN TER_BLK_SIZE_LQ_HP: IIDC at 16x16 level DM355_MPEG4E_INTRA_IN TER_BLK_SIZE_HQ_LP: IIDC at 8x8 level
SVH	XDAS_Int32	Input	0: Encode in mpeg4 mode 1: Encode in mpeg4 with short video header mode

IMP4VENC_DynamicParams

Description

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

Fields

Field	Datatype	Input/Output	Description
videncDynamicparams	IVIDENC1_DynamicParams	Input	See IVIDENC1_DynamicParams data structure for details
intraDcVlcThr	XDAS_Int32	Input	Intra VLC Threshold (0 to 7) Maximum value supported is 7 Minimum value supported is 0. Use 0 for better quality.
intraThres	XDAS_Int32	Input	Threshold for introducing Intra MBs. Maximum value supported is 0xFFFF Minimum value supported is 0. Use a value of 192 for better quality.
intraAlgo	XDAS_Int32	Input	INTRA/INTER Decision Algorithm. DM355_MPEG4E_INTRA_INTER_DECISION_LQ_HP: for low quality and high performance DM355_MPEG4E_INTRA_INTER_DECISION_HQ_LP: for high quality and low performance DM355_MPEG4E_INTRA_INTER_DECISION_HQ_LP: for better quality.
numMBRows	XDAS_Int32	Input	Number of MB rows in a Packet. Maximum value is "subWindowHeight/16". This indicates the packet size. Minimum value supported is 1.
initQ	XDAS_Int32	Input	Initial Q (at picture head). 0: automatically determined 2-31: Force initial Q This is for I frame quantization. (Default value is 3). This should have a value less than rcQ_MAX and more than rcQ_MIN.
rcQ_MAX	XDAS_Int32	Input	Q MAX value Maximum value supported is 31 Minimum value supported is 2
rcQ_MIN	XDAS_Int32	Input	Q MIN value Maximum value supported is 31 Minimum value supported is 2
rateFix	XDAS_Int32	Input	Q FIX PIC: Change Q parameter 0: at MB 1: at picture start Other values not supported.
rateFixRange	XDAS_Int32	Input	Q FIX RANGE: Q scale differential range in Q fix mode. If rateFix = 0, then this parameter is ignored. Maximum value supported is 31 Minimum value supported is 2 Other values are not supported.
InitQ_P	XDAS_Int32	Input	Initial Q (at picture head). 0: Automatically determined 2-31: Force initial Q This is for P frame quantization. (Default value is 4) **Note:** This should have a value less than rcQ_MAX and more than rcQ_MIN.

Field	Datatype	Input/Output	Description
meRange	XDAS_Int32	Input	Motion Estimation Range 7: ME7 31: ME31 Others not supported. **Note:** This parameter is not used in the current version as it is decided by meAlgo parameter
MeAlgo	XDAS_Int32	Input	Motion estimation algorithm DM355_MPEG4E_ME_LQ_HP: For lowest quality and highest performance. DM355_MPEG4E_ME_MQ_MP: For moderate quality and moderate performance. DM355_MPEG4E_ME_HQ_MP: For high quality and moderate performance. DM355_MPEG4E_ME_HQ_LP: For highest quality and lowest performance.
SkipMBAlgo	XDAS_Int32	Input	P Skip MB algorithm DM355_MPEG4E_SKIP_MB_LQ_HP: Non-Bonus Skip MB For low quality and high performance DM355_MPEG4E_SKIP_MB_HQ_LP: Bonus SKIP MB For high quality and low performance.
UMV	XDAS_Int32	Input	Unrestricted motion vector DM355_MPEG4E_UMV_LQ_HP: disable DM355_MPEG4E_UMV_HQ_LP: enable Note: If UMV is enabled, the minimum input stream width supported is 192.
IIDC	XDAS_Int32	Input	DM355_MPEG4E_INTRA_INTER_BLK_SIZE_LQ_HP: IIDC at 16x16 level DM355_MPEG4E_INTRA_INTER_BLK_SIZE_HQ_LP: IIDC at 8x8 level
MVDataEnable	XDAS_Int32	Input	0: Disable motion vector access 1: Enable motion vector access

IMP4VENC_InArgs

Description

This structure defines the input argument parameters and any other implementation specific parameters for the MPEG4 Encoder instance object. The basic input parameters are defined in XDM data structure, IVIDENC_InArgs.

Fields

Field	Datatype	Input/ Output	Description
videncInArgs	IVIDENC1_InArgs	Input	See IVIDENC1_InArgs data structure for details
subWindowHozOfst	XDAS_Int32	Input	Horizontal Offset of the Subwindow from the input image. Not supported in this version
subWindowVerOfst	XDAS_Int32	Input	Vertical Offset of the Subwindow from the input image. Not supported in this version

IMP4VENC_OutArgs

Description

This structure defines the output argument parameters and any other implementation specific parameters for the MPEG4 Encoder instance object. The basic input parameters are defined in XDM data structure, IVIDENC_OutArgs.

Fields

Field	Datatype	Input/ Output	Description
videncOutArgs	IVIDENC1_OutArgs	Input	See IVIDENC1_OutArgs data structure for details
mvDataSize	XDAS_Int32	Input	Size of the motion vector array

DM355_MPEG4E_ERROR

Description

This enum structure defines the error bit for each of the creation time and run time parameters for error reporting purposes.

Fields

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
DM355_MPEG4E_ERROR	DM355_MPEG4E_INVALID_IMAGEWIDTH	Bit 1 1 : Invalid image width 0 : Ignore
	DM355_MPEG4E_INVALID_IMAGEHEIGHT	Bit 2 1 : Invalid image height 0 : Ignore
	DM355_MPEG4E_INVALID_ENCODINGPRESET	Bit 3 1 : Invalid encoding preset 0 -Ignore
	DM355_MPEG4E_INVALID_RATECONTROLPRESET	Bit 4 1 : Invalid Rate control preset 0 -Ignore
	DM355_MPEG4E_INVALID_MAXINTERFRMINTERVAL	Bit 5 1 : Invalid maximum inter frame interval (if value is other than 1) 0 : Ignore
	DM355_MPEG4E_INVALID_INPUTCONTENTTYPE	Bit 6 1 : Invalid input content type 0 : Ignore
	DM355_MPEG4E_INVALID_RECONCHROMAFORMAT	Bit 7 1 : Invalid Recon chroma format 0 : Ignore
	DM355_MPEG4E_INVALID_FRAMERATE	Bit 8 1 : Invalid value of frame rate 0 : Ignore
	DM355_MPEG4E_INVALID_INTRAFRAMEINTERVAL	Bit 9 1 : Invalid value of intra frame interval 0 : Ignore
	DM355_MPEG4E_INVALID_INTRADCVLCTHR	Bit 10 1 : Invalid value of intra DC VLC threshold 0 : Ignore
	DM355_MPEG4E_INVALID_ROTATION	Bit 11 1 : Invalid value of rotation 0 : Ignore
	DM355_MPEG4E_INVALID_GENERATEHEADER	Bit 12 1 : Invalid value of generate header parameter 0 : Ignore
	DM355_MPEG4E_INVALID_MVDATAENABLE	Bit 14 1 : Invalid value of motion vector access parameter 0 : Ignore
	DM355_MPEG4E_INVALID_INTRATHRES	Bit 16 1 : Invalid value intra threshold 0 : Ignore
	DM355_MPEG4E_INVALID_INTRAALGO	Bit 17 1 : Invalid value of intra algo 0 : Ignore
	DM355_MPEG4E_INVALID_NUMMBROWS	Bit 18

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
		1 : Invalid value of number of MB rows 0 : Ignore
	DM355_MPEG4E_INVALID_INITQ	Bit 19 1 : Invalid initial quantization value for I frame 0 : Ignore
	DM355_MPEG4E_INVALID_RCQMAX	Bit 20 1 : Invalid value of rcMAX parameter 0 : Ignore
	DM355_MPEG4E_INVALID_RCQMIN	Bit 21 1 : Invalid value of rcMIN parameter 0 : Ignore
	DM355_MPEG4E_INVALID_RATEFIX	Bit 22 1 : Invalid value of rate fix parameter
	DM355_MPEG4E_INVALID_RATEFIXRANGE	Bit 23 1 : Invalid value of rate fix range parameter 0 : Ignore
	DM355_MPEG4E_INVALID_VBVSIZE	Bit 24 1 : Invalid value of virtual buffer verifier parameter 0 : Ignore
	DM355_MPEG4E_INVALID_INITQP	Bit 25 1 : Invalid initial quantization value for I frame 0 : Ignore
	DM355_MPEG4E_INVALID_MERANGE	Bit 26 1 : Invalid range for Motion estimation 0 : Ignore
	DM355_MPEG4E_INVALID_MEALGO	Bit 27 1 : Invalid value of ME algo parameter 0 : Ignore
	DM355_MPEG4E_INVALID_SKIPMBALGO	Bit 28 1 : Invalid value of skip MB algo parameter 0 : Ignore
	DM355_MPEG4E_INVALID_UMV	Bit 29 1 : Invalid value of UMV parameter 0 : Ignore
	DM355_MPEG4E_INVALID_IIDC	Bit 30 1 : Invalid value of IIDC parameter 0 : Ignore
	DM355_MPEG4E_INVALID_SVH	Bit 31 1 : Invalid value of SVH parameter 0 : Ignore

4.2 Interface Functions

This section describes the Application Programming Interfaces (APIs) used in the MPEG4 encoder. The APIs are logically grouped into the following categories:

- **Creation** – `algNumAlloc(), algAlloc(), dmaGetChannelCnt(), dmaGetChannels()`
- **Initialization** – `algInit(), dmaInit()`
- **Termination** – `algFree()`

You must call these APIs in the following sequence:

Step 1. `algNumAlloc()`
Step 2. `algAlloc()`
Step 3. `algInit()`
Step 4. `control()`
Step 5. `algActivate()`
Step 6. `process()`
Step 7. `algDeactivate()`
Step 8. `algFree()`

`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 the *TMS320 DSP Algorithm Standard API Reference* (SPRU360).

4.2.1 Creation APIs

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

Note: For more details on the External Data Memory requirements, see the *MPEG4 Simple Profile Encoder Data Sheet* (SPRS488).

4.2.2 Initialization API

The initialization API initializes an instance of the algorithm. The initialization parameters are defined in the Params structure (see the Data Structures section for details).

The following code is an example of initializing the Params structure and creating an encoder instance with base parameters.

```
{
    ....................................

    IVIDDEC2_Params        params;

    // Set the create time base parameters
    params.size = sizeof(IVIDDEC2_Params);
    params.maxHeight = 480;
    params.maxWidth = 720;
    params.maxFrameRate = XDM_DEFAULT;
    params.maxBitRate = BITRATE;
    params.dataEndianness = XDM_BYTE;
    params.maxInterFrameInterval = XDM_DEFAULT;
    params.inputChromaFormat = XDM_YUV_422ILE;
    params.inputContentType = IVIDEO_PROGRESSIVE;
    params.reconChromaFormat = XDM_DEFAULT;

   handle = (IALG_Handle) ALG_create((IALG_Fxns *)& MP4VENC_TI_IMP4VENC,
                                                (IALG_Handle) NULL,
                                                (IALG_Params *) &params)
.....................................
}
```

The following code is an example of initializing the Params structure and creating an instance with extended parameters.

```
{
    ....................................

    VIDENC1_Params            params;
    IMP4VENC_Params        extParams;

    // Set the create time base parameters
    params.size = sizeof(IMP4VENC_Params);
    params.encodingPreset = XDM_USER_DEFINED;
    params.rateControlPreset = IVIDEO_STORAGE;
    params.maxHeight = 480;
    params.maxWidth = 720;
    params.maxFrameRate = FRAMERATE;
    params.maxBitRate = BITRATE;
    params.dataEndianness = XDM_BYTE;
    params.maxInterFrameInterval = XDM_DEFAULT;
    params.inputChromaFormat = XDM_YUV_422ILE;
    params.inputContentType = IVIDEO_PROGRESSIVE;
    params.reconChromaFormat = XDM_DEFAULT;

    // Set the create time extended parameters

    extParams.videncParams = params;

    extParams.subWindowHeight = 480;
    extParams.subWindowWidth = 720;
    extParams.intraFrameInterval = 30;
    extParams.intraDcVlcThr = XDM_DEFAULT;
    extParams.rotation = XDM_DEFAULT;
    extParams.intraThres = 192;
    extParams.intraAlgo = 1;
    extParams.numMBRows = 5;

    extParams.initQ = 3;
    extParams.rcQ_MAX = 31;
    extParams.rcQ_MIN = 1;
    extParams.rateFix = XDM_DEFAULT;
    extParams.rateFixRange = 4;
    extParams.VBV_size = 10000;
    extParams.InitQ_P = 4;

    extParams.meRange = 31;
    extParams.meAlgo = 1;
    extParams.SkipMBAlgo = XDM_DEFAULT;
    extParams.UMV = XDM_DEFAULT;
    extParams.IIDC = XDM_DEFAULT;

   handle = (IALG_Handle) ALG_create((IALG_Fxns *)& MP4VENC_TI_IMP4VENC,
                                     (IALG_Handle) NULL,
                                     (IALG_Params *) & extParams)
.....................................
}
```

4.2.3 Control Processing API

The control API is used before a call to `process()` to enquire about the number and size of I/O buffers, or to set the dynamic params, or get status of encoding. The following code gives an example for initializing the base dynamic parameters for a 720x480 stream.

```
{
.....................................

    VIDENC1_DynamicParams        dynParams;
    IVIDENC1_Status        status;
.....................................

    // Set the dynamic base parameters
    dynParams.size = sizeof(VIDENC1_DynamicParams);
    dynParams.inputHeight = 480;
    dynParams.inputWidth = 720;
```

```
    dynParams.refFrameRate = FRAMERATE;
    dynParams.targetFrameRate = FRAMERATE;
    dynParams.targetBitRate = BITRATE;
    dynParams.intraFrameInterval = 30;
    dynParams.generateHeader = XDM_DEFAULT;
    dynParams.captureWidth = XDM_DEFAULT;
    dynParams.forceFrame = XDM_DEFAULT;
    dynParams.interFrameInterval = XDM_DEFAULT;
    dynParams.mbDataFlag = XDM_DEFAULT;



/* Set Dynamic Params */
retVal = ividEncfxns->control((IVIDENC1_Handle)handle, XDM_SETPARAMS,
                    (IVIDENC1_DynamicParams *)& dynParams,
                    (IVIDENC1_Status *)&status);
.....
  }
```

The following code gives an example for initializing the extended dynamic parameters for a 720x480 stream without motion vector access.

```
{
.....
    VIDENC1_DynamicParams    dynParams;
    IVIDENC1_Status        status;
    IMP4VENC_DynamicParams    extDynParams;
.....

    // Set the dynamic base parameters
    dynParams.size = sizeof(IMP4VENC_Params);
    dynParams.inputHeight = 480;
    dynParams.inputWidth = 720;
    dynParams.refFrameRate = FRAMERATE;
    dynParams.targetFrameRate = FRAMERATE;
    dynParams.targetBitRate = BITRATE;
    dynParams.intraFrameInterval = 30;
    dynParams.generateHeader = XDM_DEFAULT;
    dynParams.captureWidth = XDM_DEFAULT;
    dynParams.forceFrame = XDM_DEFAULT;
    dynParams.interFrameInterval = XDM_DEFAULT;
    dynParams.mbDataFlag = XDM_DEFAULT;

    // Set the extended dynamic parameters
    extDynParams.videncDynamicParams = dynParams;

    extDynParams.intraDcVlcThr = XDM_DEFAULT;
    extDynParams.intraThres = 192;
    extDynParams.intraAlgo = 1;
    extDynParams.numMBRows = 5;
    extDynParams.initQ = 3;
    extDynParams.rcQ_MAX = 31;
    extDynParams.rcQ_MIN = 1;
    extDynParams.rateFix = XDM_DEFAULT;
    extDynParams.rateFixRange = 4;
    extDynParams.InitQ_P = 4;
    extDynParams.meRange = 31;
    extDynParams.meAlgo = 1;
    extDynParams.SkipMBAlgo = XDM_DEFAULT;
    extDynParams.UMV = XDM_DEFAULT;
    extDynParams.IIDC = XDM_DEFAULT;

    extDynParams.MVDataEnable = XDM_DEFAULT;

/* Set Dynamic Params */
retVal = ividEncfxns->control((IVIDENC1_Handle)handle, XDM_SETPARAMS,
                    (IVIDENC1_DynamicParams *)& extDynParams,
                    (IVIDENC1_Status *)&status);
.....
  }
```

4.2.4 Data Processing API

The data processing API processes the input data. The following sample code gives an example of process call.

```
{
.....................................................

retVal = ividEncfxns->process((IVIDENC1_Handle) handle,
                              (IVIDEO1_BufDescIn *) &inputBufDesc,
                              (XDM_BufDesc *) &outputBufDesc,
                              (IVIDENC1_InArgs *) &inArgs,
                             (IVIDENC1_OutArgs *) &outArgs);
.....................................................
}
```

4.2.5 Termination API

The termination API terminates the algorithm instance and frees up the memory space that it uses.

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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