MPEG4 Simple Profile Encoder on DM6446

User's Guide

![Texas Instruments]

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

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

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2009, Texas Instruments Incorporated

Read This First

About This Manual

This document describes how to install and work with Texas Instruments' (TI) MPEG4 Simple Profile Encoder implementation on the DM6446 platform. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

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

Intended Audience

This document is intended for system engineers who want to integrate TI's codecs with other software to build a multimedia system based on the DM6446 platform.

This document assumes that you are fluent in the C language, have a good working knowledge of Digital Signal Processing (DSP), digital signal processors, and DSP applications. Good knowledge of eXpressDSP Algorithm Interface Standard (XDAIS) and eXpressDSP Digital Media (XDM) standard will be helpful.

How to Use This Manual

This document includes the following chapters:

❑ **Chapter 1 - Introduction**, provides a brief introduction to the XDAIS and XDM standards. It also provides an overview of the codec and lists its supported features.

❑ **Chapter 2 - Installation Overview**, describes how to install, build, and run the codec.

❑ **Chapter 3 - Sample Usage**, describes the sample usage of the codec.

❑ **Chapter 4 - API Reference**, describes the data structures and interface functions used in the codec.

❑ **Appendix A - Motion Vector Access API,** describes the Motion Vector Access API which is used by the application to encode a frame.

❑ **Appendix B - Revision History,** highlights the changes made to the SPRUEA2B codec specific user guide to make it SPRUEA2C.

Related Documentation From Texas Instruments

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

❑ *TMS320 DSP Algorithm Standard Rules and Guidelines* (literature number SPRU352) defines a set of requirements for DSP algorithms that, if followed, allow system integrators to quickly assemble production-quality systems from one or more such algorithms.

❑ *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360) describes all the APIs that are defined by the TMS320 DSP Algorithm Interface Standard (also known as XDAIS) specification.

❑ *Technical Overview of eXpressDSP - Compliant Algorithms for DSP Software Producers* (literature number SPRA579) describes how to make algorithms compliant with the TMS320 DSP Algorithm Standard which is part of TI's eXpressDSP technology initiative.

❑ *Using the TMS320 DSP Algorithm Standard in a Static DSP System* (literature number SPRA577) describes how an eXpressDSP-compliant algorithm may be used effectively in a static system with limited memory.

❑ *DMA Guide for eXpressDSP-Compliant Algorithm Producers and Consumers* (literature number SPRA445) describes the DMA architecture specified by the TMS320 DSP Algorithm Standard (XDAIS). It also describes two sets of APIs used for accessing DMA resources: the IDMA2 abstract interface and the ACPY2 library.

❑ *eXpressDSP Digital Media (XDM) Standard API Reference* (literature number SPRUEC8)

The following documents describe TMS320 devices and related support tools:

❑ *Design and Implementation of an eXpressDSP-Compliant DMA Manager for C6X1X* (literature number SPRA789) describes a C6x1x-optimized (C6211, C6711) ACPY2 library implementation and DMA Resource Manager.

❑ *TMS320C64x+ Megamodule* (literature number SPRAA68) describes the enhancements made to the internal memory and describes the new features which have been added to support the internal memory architecture's performance and protection.

❑ *TMS320C64x+ DSP Megamodule Reference Guide* (literature number SPRU871) describes the C64x+ megamodule peripherals.

❑ *TMS320C64x to TMS320C64x+ CPU Migration Guide* (literature number SPRAA84) describes migration from the Texas Instruments TMS320C64x™ digital signal processor (DSP) to the TMS320C64x+™ DSP.

❑ *TMS320C6000 Optimizing Compiler v 6.0 Beta User's Guide* (literature number SPRU187N) explains how to use compiler tools such as compiler, assembly optimizer, standalone simulator, library-build utility, and C++ name demangler.

❑ *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (literature number SPRU732) describes the CPU architecture, pipeline, instruction set, and interrupts of the C64x and C64x+ DSPs.

❑ TMS320DM6446 Digital Media System-on-Chip (literature number SPRS283)

❑ *TMS320DM6446 Digital Media System-on-Chip Errata (Silicon Revision 1.0)* (literature number SPRZ241) describes the known exceptions to the functional specifications for the TMS320DM6446 Digital Media System-on-Chip (DMSoC).

❑ TMS320DM6443 Digital Media System-on-Chip (literature number SPRS282)

❑ *TMS320DM6443 Digital Media System-on-Chip Errata (Silicon Revision 1.0)* (literature number SPRZ240) describes the known exceptions to the functional specifications for the TMS320DM6443 Digital Media System-on-Chip (DMSoC).

❑ *TMS320DM644x DMSoC DSP Subsystem Reference Guide* (literature number SPRUE15) describes the digital signal processor (DSP) subsystem in the TMS320DM644x Digital Media System-on-Chip (DMSoC).

❑ *TMS320DM644x DMSoC ARM Subsystem Reference Guide* (literature number SPRUE14) describes the ARM subsystem in the TMS320DM644x Digital Media System on a Chip (DMSoC).

❑ *DaVinci Technology - Digital Video Innovation Product Bulletin* (Rev. A) (literature number SPRT378A)

❑ *The DaVinci Effect: Achieving Digital Video Without Complexity White Paper* (literature number SPRY079)

❑ *DaVinci Benchmarks Product Bulletin* (literature number SPRT379)

❑ *DaVinci Technology for Digital Video White Paper* (literature number SPRY067)

❑ *The Future of Digital Video White Paper* (literature number SPRY066)

Related Documentation

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

❑ *ISO/IEC 11172-2 Information Technology -- Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbits/s -- Part 2: Video (MPEG-1 video standard)*

❑ *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*

Abbreviations

The following abbreviations are used in this document.

Table 1-1. 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

Abbreviation	Description
QVGA	Quarter Video Graphics Array
SP	Simple Profile
SQCIF	Sub Quarter Common Intermediate Format
TM5	Test Model 5
TMN	Test Model Near term
TMN5	Test Model Near term, Version 5
VBR	Variable Bit Rate
VBV	Video rate Buffer Verifier
VM4	Verification Model 4
VOL	Video Object Layer
VOP	Video Object Plane
VOS	Video Object Sequence
XDAIS	eXpressDSP Algorithm Interface Standard
XDM	eXpressDSP Digital Media

Text Conventions

The following conventions are used in this document:

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

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

Product Support

When contacting TI for support on this codec, quote the product name (MPEG4 Simple Profile Encoder on DM6446) and version number. The version number of the codec is included in the title of the release notes that accompanies this codec.

Trademarks

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

All trademarks are the property of their respective owners.

Contents

Figures

This page is intentionally left blank

Tables

This page is intentionally left blank

Introduction

This chapter provides a brief introduction to XDAIS and XDM. It also provides an overview of TI's implementation of the MPEG4 Simple Profile Encoder on the DM6446 platform and its supported features.

1.1 Overview of XDAIS and XDM

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).

1.1.1 XDAIS Overview

An eXpressDSP-compliant algorithm is a module that implements the abstract interface IALG. The IALG API takes the memory management function away from the algorithm and places it in the hosting framework. Thus, an interaction occurs between the algorithm and the framework. This interaction allows the client application to allocate memory for the algorithm and also share memory between algorithms. It also allows the memory to be moved around while an algorithm is operating in the system. In order to facilitate these functionalities, the IALG interface defines the following APIs:

❑ `algAlloc()`

❑ `algInit()`

❑ `algActivate()`

❑ `algDeactivate()`

❑ `algFree()`

The `algAlloc()` API allows the algorithm to communicate its memory requirements to the client application. The `algInit()` API allows the algorithm to initialize the memory allocated by the client application. The `algFree()` API allows the algorithm to communicate the memory to be freed when an instance is no longer required.

Once an algorithm instance object is created, it can be used to process data in real-time. The `algActivate()` API provides a notification to the algorithm instance that one or more algorithm processing methods is about to be run zero or more times in succession. After the processing methods have been run, the client application calls the `algDeactivate()` API prior to reusing any of the instance's scratch memory.

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

1.1.2 XDM Overview

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

(For example audio, video, image, and speech). The XDM standard defines the following two APIs:

❑   control()

❑   process()

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

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.
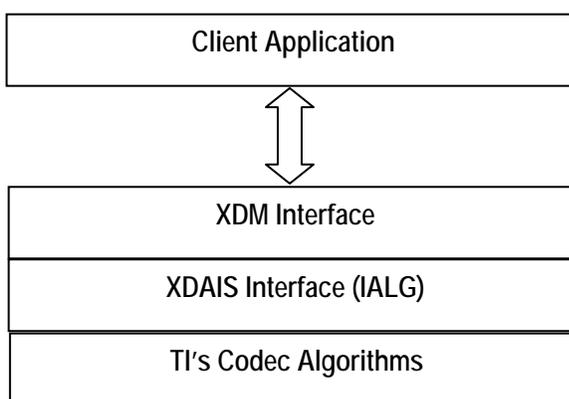
As depicted in the figure, XDM is an extension to XDAIS and forms an interface between the client application and the codec component. XDM insulates the client application from component-level changes. Since TI's multimedia algorithms are XDM compliant, it provides you with the flexibility to use any 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   Overview of MPEG4 Simple Profile Encoder

MPEG4 is the ISO/IEC recommended standard for video compression. Figure 1-1 shows the working of the MPEG4 Encoder.
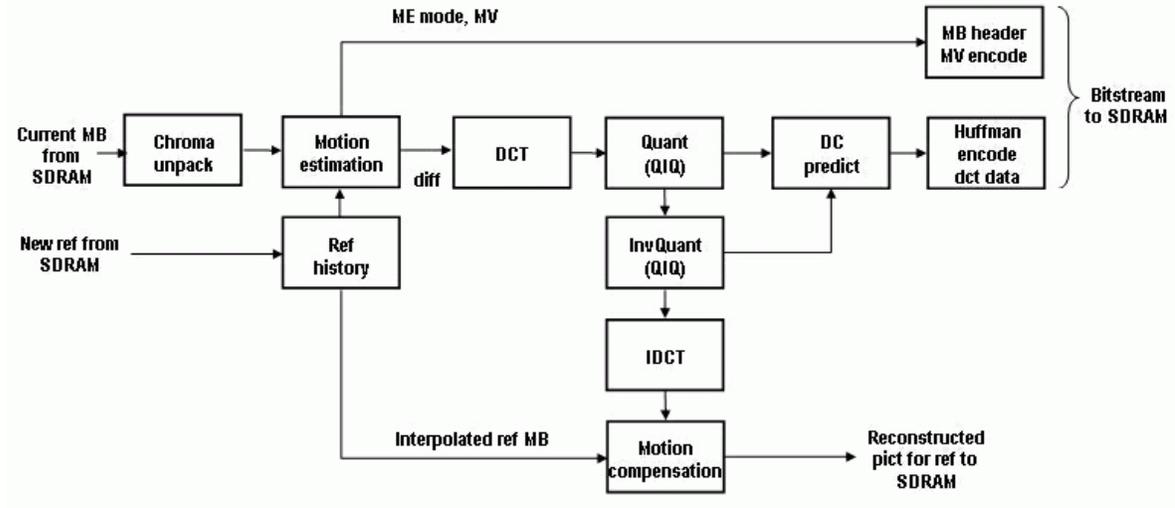
Figure 1-1. Working of MPEG4 Video Encoder

An input video sequence for a MPEG4 Encoder consists of frames. MPEG4 Encoder accepts the input frames in YUV format. Video coding aims at providing a compact representation of the information in the video frames by removing spatial redundancies that exist within the frames and temporal redundancies that exist between successive frames.

The MPEG4 standard is based on Discrete Cosine Transform (DCT) to remove spatial redundancies. Motion estimation and motion compensation is used to remove temporal redundancies.

All frames in a video sequence are categorized as I-frames or 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 depends on information from a previous frame for 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 will be much more efficient to encode than encoding the original frame.

A more sophisticated approach to increase encoding 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.

From this point onwards, all references to MPEG4 Encoder means MPEG4 Simple Profile Encoder only.

1.3 Supported Services and Features

This user guide accompanies TI's implementation of MPEG4 Encoder on the DM6446 platform.

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

❑   Compliant with the MPEG4 Simple Profile levels 0, 1, 2, 3, 4A, and 5

❑   Supports H.263 Baseline Profile levels 10, 20, 30, and 45

❑   Generates bit-streams that are compliant with the Video Buffering Verifier as per MPEG4 standard

❑   Supports Data Partitioning (DP) and Reversible Variable Length Code (RVLC)

❑   Supports AC prediction

❑   Supports Adaptive and Mandatory Intra refresh

❑   Supports image width and height which are non-multiple of 16

❑   Supports Unrestricted Motion Vectors (UMV) for both MPEG4 and H.263 Annexure D. UMV can be switched on/off at run-time

❑   Supports addition of Video Sequence End code in the bit-stream

❑   Motion estimation is performed using the IMCOP hardware accelerator provided on the VICP

The other explicit features that TI's MPEG4 Encoder provides are:

❑   Supports TI's proprietary rate control algorithms

❑   Supports TI's proprietary content adaptive motion estimation

❑   Supports resolutions up to PAL D1(720 x 576), including standard image sizes such as SQCIF, QCIF, CIF, QVGA,VGA, and D1

❑   Supports Half Pel Interpolation (HPI) for motion estimation

❑ Supports setting of Quantization Parameter (QP) for I-frames and P-frames

❑ Supports I-frame insertion and changing the size of video packets at run-time

❑ Supports 422i or 420 input formats for the frames

❑ Supports only header encoding (VOL Header)

❑ Supports motion vector access

❑ Supports capture width

❑ Provides high quality options using encoding preset

❑ eXpressDSP Digital Media (XDM 1.0 IVIDENC1) compliant

Encoder does not support interlaced input hence quality degrades when interlaced input is provided. For better quality, use software based de-interlacer and provide progressive input to the encoder.

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	2-2
2.2 Installing the Component	2-2
2.3 Before Building the Sample Test Application	2-4
2.4 Building and Running the Sample Test Application	2-5
2.5 Configuration Files	2-6
2.6 Standards Conformance and User-Defined Inputs	2-9
2.7 Uninstalling the Component	2-9
2.8 Evaluation Version	2-9

2.1 System Requirements

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

2.1.1 Hardware

This codec has been built and tested on the DM6446 EVM with XDS560 JTAG emulator.

2.1.2 Software

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

❑ **Development Environment:** This project is developed using Code Composer Studio version 3.2.37.12.

❑ **Code Generation Tools:** This project is compiled, assembled, archived, and linked using the code generation tools version 6.0.8.

2.2 Installing the Component

The codec component is released as a compressed archive. To install the codec, extract the contents of the zip file to your local hard disk. The zip file extraction creates a top-level directory called 100_V_MPEG4_E_2_00, under which another directory named DM6446_SP_001 is created.

Figure 2-2 shows the sub-directories created in the DM6446_SP_001 directory.
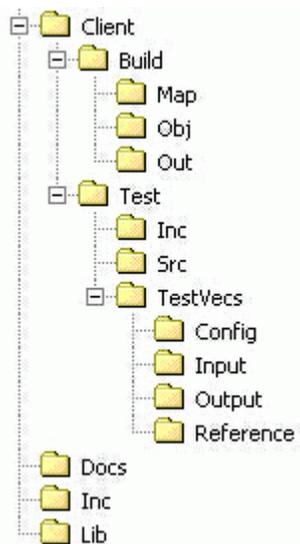
Figure 2-2. Component Directory Structure

> **Note:**
>
> If you are installing an evaluation version of this codec, the directory name will be 100E_V_MPEG4_E_2_00.

Table 2-1 provides a description of the sub-directories created in the DM6446_SP_001 directory.

Table 2-1. Component Directories

Sub-Directory	Description
\Inc	Contains XDM related header files which allow interface to the codec library
\Lib	Contains the codec library file
\Docs	Contains user guide and datasheet
\Client\Build	Contains the sample test application project (.pjt) file
\Client\Build\Map	Contains the memory map generated on compilation of the code
\Client\Build\Obj	Contains the intermediate .asm and/or .obj file generated on compilation of the code
\Client\Build\Out	Contains the final application executable (.out) file generated by the sample test application
\Client\Test\Src	Contains application C files
\Client\Test\Inc	Contains header files needed for the application code
\Client\Test\TestVecs\Input	Contains input test vectors
\Client\Test\TestVecs\Output	Contains output generated by the codec
\Client\Test\TestVecs\Reference	Contains read-only reference output to be used for verifying against codec output
\Client\Test\TestVecs\Config	Contains configuration parameter files

2.3 Before Building the Sample Test Application

This codec is accompanied by a sample test application. To run the sample test application, you need DSP/BIOS and TI Framework Components (FC).

This version of the codec has been validated with DSP/BIOS version 5.32.02 and Framework Component (FC) version 2.20.00.15.

2.3.1 Installing DSP/BIOS

You can download DSP/BIOS from the TI external website:

https://www-a.ti.com/downloads/sds_support/targetcontent/bios/index.html

Install DSP/BIOS at the same location where you have installed Code Composer Studio. For example:

<install directory>\CCStudio_v3.2

The sample test application uses the following DSP/BIOS files:

❑ Header file, bcache.h available in the
<install directory>\CCStudio_v3.2\<bios_directory>\packages
\ti\bios\include directory.

❑ Library file, biosDM420.a64P available in the
<install directory>\CCStudio_v3.2\<bios_directory>\packages
\ti\bios\lib directory.

2.3.2 Installing Framework Component (FC)

You can download FC from the TI external website:

https://www-a.ti.com/downloads/sds_support/targetcontent/FC/index.html

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

<install directory>\CCStudio_v3.2

The test application uses the following DMAN3 files:

❑ Library file, dman3.a64P available in the
<install directory>\CCStudio_v3.2\<fc_directory>\packages
\ti\sdo\fc\dman3 directory.

❑ Header file, dman3.h available in the
<install directory>\CCStudio_v3.2\<fc_directory>\packages
\ti\sdo\fc\dman3 directory.

❑ Header file, idma3.h available in the
<install directory>\CCStudio_v3.2\<fc_directory>\fctools\packages
\ti\xdais directory.

2.4   Building and Running the Sample Test Application

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

1) Verify that you have installed TI's Code Composer Studio version 3.2.37.12 and code generation tools version 6.0.8.

2) Verify that the codec object library mp4venc_ti.l64P exists in the \Lib sub-directory.

3) Open the test application project file, TestAppEncoder.pjt in Code Composer Studio. This file is available in the \Client\Build sub-directory.

4) Select **Project > Build** to build the sample test application. This creates an executable file, TestAppEncoder.out in the \Client\Build\Out sub-directory.

5) Select **File > Load**, browse to the \Client\Build\Out sub-directory, select the codec executable created in step 4, and load it into Code Composer Studio in preparation for execution.

6) Select **Debug > Run** to execute the sample test application.

   The sample test application takes the input files stored in the \Client\Test\TestVecs\Input sub-directory, runs the codec, and uses the reference files stored in the \Client\Test\TestVecs\Reference sub-directory to verify that the codec is functioning as expected.

7) On successful completion, the application displays one of the following messages for each frame:

   o   "Encoder compliance test failed for this frame" or "Encoder compliance test passed for this configuration " (if x is 1)

   o   "Encoder output dump completed" (if x is 0)

2.5 Configuration Files

This codec is shipped along with:

❑ Generic configuration file (TestVecs.cfg) – specifies input and reference files for the sample test application.

❑ Encoder configuration file (Testparams.cfg) – specifies the configuration parameters used by the test application to configure the Encoder.

2.5.1 Generic Configuration File

The sample test application shipped along with the codec uses the configuration file, TestVecs.cfg for determining the input and reference files for running the codec and checking for compliance. The TestVecs.cfg file is available in the \Client\Test\TestVecs\Config sub-directory.

The format of the TestVecs.cfg file is:

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

where:

❑ X may be set as:

    ○ 1 - for compliance checking, no output file is created

    ○ 0 - for writing the output to the output file

❑ `Config` is the Encoder configuration file. For details, see Section 2.5.2.

❑ `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).

A sample TestVecs.cfg file is as shown:

```
0
..\..\Test\TestVecs\Config\Testparams.cfg
..\..\Test\TestVecs\Input\foreman_vga_422.yuv
..\..\Test\TestVecs\Output\foreman_vga_422.bits
```

2.5.2 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.

```
# Input File Format is as follows
# <ParameterName> = <ParameterValue> # Comment
#
##########################################################
Parameters
##########################################################
EncodingPreset      = 1              # 0:XDM_DEFAULT,
                                       1: XDM_HIGH_QUALITY,
                                       2: XDM_HIGH_SPEED,
                                       3: XDM_USER_DEFINED
ImageWidth          = 640            # Image width in Pels
ImageHeight         = 480            # Image height in Pels
FrameRate           = 30000          # Frame Rate per
                                       second*1000 (1-30)
Bitrate             = 4000000        # Bitrate(bps) #should be
                                       >= 1
ChromaFormat        = 4              # 1 => XDM_YUV_420P,
                                       3 => XDM_YUV_422IBE,
                                       4 => XDM_YUV_422ILE
IntraPeriod         = 30             # Period of I-
#Frames(Insertion of I-frame when changed runtime)
FramesToEncode      = 10             # Number of frames to be
                                       coded
RateControlPreset   = 5              # rateControlPreset,
                                       1: LOW_DELAY,
                                       2: STORAGE,
                                       3: TWOPASS,
                                       4: NONE,
                                       5:USER_DEFINED
EncodeMode          = 1              # Encoding mode,
                                       0: H.263 mode,
                                       1 : MPEG4 mode
LevelIdc            = 5              # Profile level
                                       indication for MPEG4,
                                       set to a minimum value
                                       of zero or a maximum
                                       value of 5 based on
                                       MPEG4 standard
RateControlMethod   = 4              # Rate Control Method,
                                       0:disable rate control,
                                       1-TM5,
                                       2-Not supported,
                                       3-PLR1,
                                       4-PLR3,
                                       5-Not supported,
                                       6:Not supported,
                                       7:Constrained VBR
                                       8:PLR4
VBVBufSize          = 112            # vbv_buffer_size (in
                                       multiples of 16 kbits)
MaxDelay            = 300            # Maximum allowable delay
                                       (Only applicable for
                                       RateControlMethod= 7
                                       ignored for other RC
                                       methods)
UseVOS              = 0              # VOS header insertion,
                                       0 = off,
                                       1 = on
UseGOV              = 0              # GOV header insertion,
                                       0 = off,
```

```
                                               1 = on
DPEnable              = 0                # Data partioning,
                                          0 = off,
                                          1 = on
RVLCEnable            = 0                # RVLC,
                                           0 = off,
                                           1 = on,
ResyncInterval       = 2000             # Insert resync marker
                                          (RM) after given
                                          specified bits,
                                          0 implies do not insert
HECInterval          = 0                # Insert HEC after given
                                           specified packets
                                           insertion, 0 means do
                                           not insert
AIRRate              = 0                # Adaptive intra refresh
                                           rate in MB's per frame
MIRRate              = 0                # Mandatory intra refresh
                                           rate in MB's per frame
QpIntra              = 8                # Default QP for
                                           I frame : QPI,
                                           range 1 to 31
QpInter              = 8                # Default QP for
                                           P frame : QPI,
                                           range 1 to 31
Fcode                = 5                # f_code: ME search
                                           area = 1<<f_code-1,
                                           set to 1 in case of
                                           H.263
UseHpi               = 1                # Half pixel
                                           interpolation,
                                           1 : on,
                                           Must be always on
UseAcPred            = 0                # AC prediction
                                           enable/disable
                                           0: off,
                                           1 = on
LatFrameFlag         = 0                # Last frame flag,
                                           1 = Last Frame,
                                           0 = Not Last Frame.
                                           This need to be
                                           dynamically set: only
                                           for the last frame
EncodeHeaderOnly     = 0                # Generate only header,
                                           1 = generates VOL
                                               header,
                                           0 = Normal Encoding
EnableMVAccess       = 1                # Enable Motion vector
                                          access,
                                          0: Disable,
                                          1: Enable
EnableUMV            = 1                #Enable UMV flag
                                           0:Disable
                                           1:Enable
captureWidth  = 850                     # For PAL D1(Greater >=
imagewidth with even value).
```

Any field in the IVIDENC1_Params structure (see Section 4.2.1.9) can be set in the Testparams.cfg file using the syntax shown above. If you specify additional fields in the Testparams.cfg file, ensure to modify the test application appropriately to handle these fields.

2.6 Standards Conformance and User-Defined Inputs

To check the conformance of the codec for the default input file shipped along with the codec, follow the steps as described in Section 2.4.

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

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

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

3) Edit the configuration file, TestVecs.cfg available in the \Client\Test\TestVecs\Config sub-directory. For details on the format of the TestVecs.cfg file, see Section 2.5.1.

4) Execute the sample test application. On successful completion, the application displays one of the following messages for each frame:

   o "Encoder compliance test failed for this frame" or "Encoder compliance test passed for this configuration " (if x is 1)

   o "Encoder output dump completed" (if x is 0)

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

2.7 Uninstalling the Component

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

2.8 Evaluation Version

If you are using an evaluation version of this codec, a Texas Instruments logo will be visible in the output.

Note:

Compliance test succeeds only for sample input file, provided with evaluation package , due to presence of Texas instruments logo in the encoded file. It should not be checked for other inputs.

This page is intentionally left blank

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 IMP4VENC extended and base class of the MPEG4 Encoder library. The main test application files are TestAppEncoder.c and TestAppEncoder.h. These files are available in the \Client\Test\Src and \Client\Test\Inc sub-directories respectively.

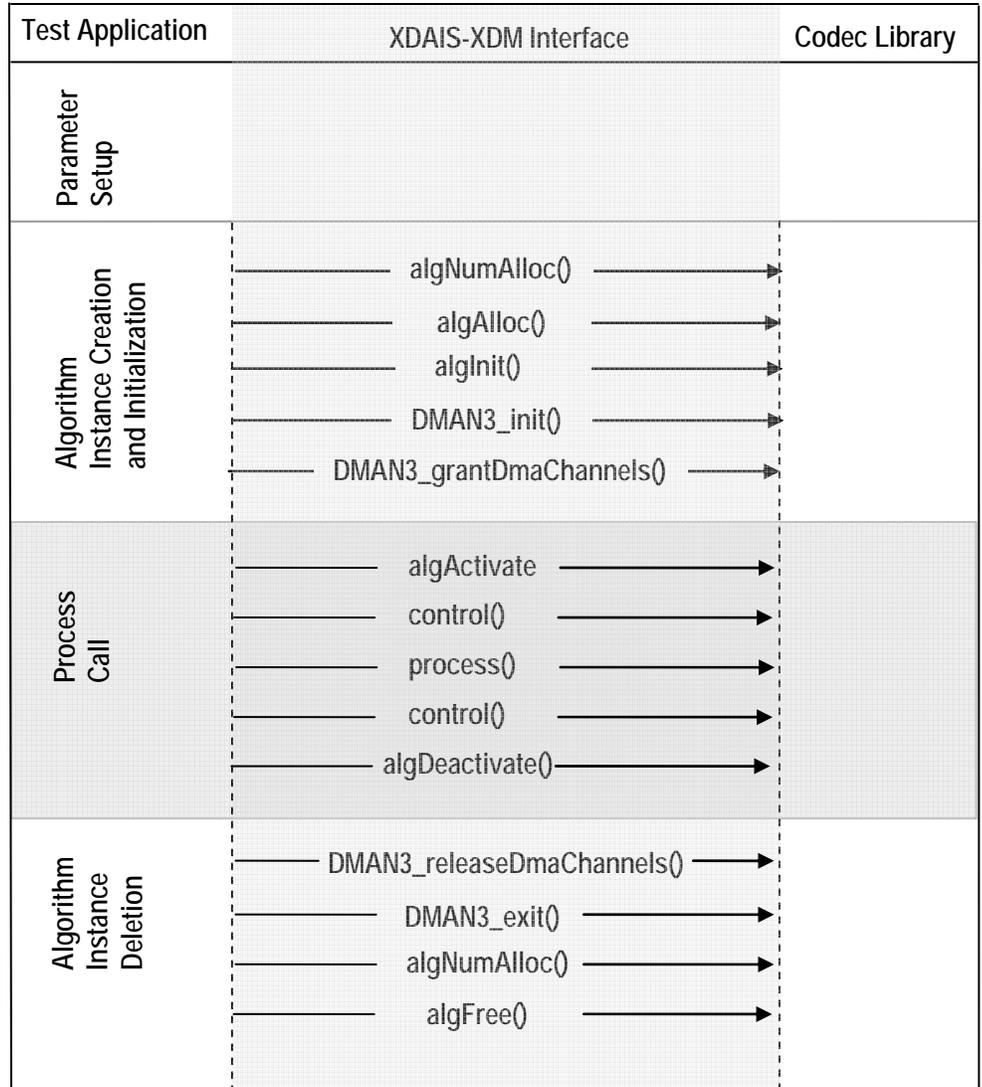Figure 3-1 depicts the sequence of APIs exercised in the sample test application.

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

3.1.1 *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 performs 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 2.5.

3) Sets the `IMP4VENC_Params` structure based on the values it reads from the Testparams.cfg file.

4) Initializes the various DMAN3 parameters.

5) Reads the input YUV frame into the application input buffer.

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 sequence:

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

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

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

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

After successful creation of the algorithm instance, the test application does DMA resource allocation for the algorithm. This requires initialization of DMA Manager Module and grant of DMA resources. This is implemented by calling DMAN3 interface functions in the following sequence:

1) `DMAN3_init()` - To initialize the DMAN module.

2) `DMAN3_grantDmaChannels()` - To grant the DMA resources to the algorithm instance.

Note:

DMAN3 function implementations are provided in dman3.a64P library.

3.1.3 *Process Call*

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

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

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

3) Calls the `process()` function to encode/decode a single frame of data. The behavior of the algorithm can be controlled using various dynamic parameters (see Section 4.2.1.10). The inputs to the process function are input and output buffer descriptors, pointer to the `IVIDENC1_InArgs` and `IVIDENC1_OutArgs` structures.

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

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

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

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

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

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

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

exhausts. It also protects the `process()` call from file operations by placing appropriate calls for cache operations I. The test application does a cache invalidate for the valid input buffers before `process()` and a cache write back invalidate for output buffers after `process()`.

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 must release the DMA channels granted by the DMA Manager interface and delete the current algorithm instance. The following APIs are called in sequence:

1) `DMAN3_releaseDmaChannels()` - To remove logical channel resources from an algorithm instance.

2) `DMAN3_exit()` - To free DMAN3 memory resources.

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

4) `algFree()` - To query the algorithm to get the memory record information.

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

This page is intentionally left blank

API Reference

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

4.1 Symbolic Constants and Enumerated Data Types

This section summarizes all the symbolic constants specified as either #define macros and/or enumerated C data types. For each symbolic constant, the semantics or interpretation of the same is also provided.

Table 4-1. List of Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IVIDEO_FrameType	IVIDEO_NA_FRAME	Frame type not available.
	IVIDEO_I_FRAME	Intra coded frame
	IVIDEO_P_FRAME	Forward inter coded frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_B_FRAME	Bi-directional inter coded frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_IDR_FRAME	Intra coded frame that can be used for refreshing video content. Not supported in this version of MPEG4 Encoder.
	IVIDEO_II_FRAME	Interlaced frame, both fields are I frames. Not supported in this version of MPEG4 Encoder.
	IVIDEO_IP_FRAME	Interlaced frame, first field is an I frame, second field is a P frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_IB_FRAME	Interlaced frame, first field is an I frame, second field is a B frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_PI_FRAME	Interlaced frame, first field is a P frame, second field is an I frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_PP_FRAME	Interlaced frame, both fields are P frames. Not supported in this version of MPEG4 Encoder.
	IVIDEO_PB_FRAME	Interlaced frame, first field is a P frame, second field is a B frame. Not supported in this version of MPEG4 Encoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_BI_FRAME	Interlaced frame, first field is a B frame, second field is an I frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_BP_FRAME	Interlaced frame, first field is a B frame, second field is a P frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_BB_FRAME	Interlaced frame, both fields are B frames. Not supported in this version of MPEG4 Encoder.
	IVIDEO_MBAFF_I_FRAME	Intra coded MBAFF frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_MBAFF_P_FRAME	Forward inter coded MBAFF frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_MBAFF_B_FRAME	Bi-directional inter coded MBAFF frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_MBAFF_IDR_FRAME	Intra coded MBAFF frame that can be used for refreshing video content. Not supported in this version of MPEG4 Encoder.
	IVIDEO_FRAMETYPE_DEFAULT	Default value is set to IVIDEO_I_FRAME
IVIDEO_ContentType	IVIDEO_CONTENTTYPE_NA	-1, Frame type is not available.
	IVIDEO_PROGRESSIVE	0, Progressive frame
	IVIDEO_PROGRESSIVE_FRAME	Is equal to IVIDEO_PROGRESSIVE
	IVIDEO_INTERLACED	1, Interlaced frame. Not supported in this version of MPEG4 Encoder.
	IVIDEO_INTERLACED_FRAME	IVIDEO_INTERLACED. Not supported in this version of MPEG4 Encoder.
	IVIDEO_INTERLACED_TOPFIELD	2, Interlaced picture, top field. Not supported in this version of MPEG4 Encoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_INTERLACED_BO TTOMFIELD	3, Interlaced picture, bottom field. Not supported in this version of MPEG4 Encoder.
	IVIDEO_CONTENTTYPE_D EFAULT	IVIDEO_PROGRESSIVE
IVIDEO_RateControlPreset	IVIDEO_NONE	No rate control is used
	IVIDEO_LOW_DELAY	Constant Bit Rate (CBR) control for video conferencing (default value.)
	IVIDEO_STORAGE	Variable Bit Rate (VBR) control for local storage (DVD) recording.
	IVIDEO_TWOPASS	Two pass rate control for non real time applications. Not supported in this version of MPEG4 Encoder.
	IVIDEO_USER_DEFINED	User defined configuration using advanced parameters (extended parameters).
	IVIDEO_RATECONTROLPR ESET_DEFAULT	IVIDEO_LOW_DELAY is default value
IVIDEO_SkipMode	IVIDEO_FRAME_ENCODED	Input content encoded
	IVIDEO_FRAME_SKIPPED	Input content skipped, that is, not encoded.
	IVIDEO_SKIPMODE_DEFA ULT	IVIDEO_FRAME_ENCODED is default
XDM_DataFormat	XDM_BYTE	Big endian stream
	XDM_LE_16	16-bit little endian stream. Not supported in this version of MPEG4 Encoder.
	XDM_LE_32	32-bit little endian stream. Not supported in this version of MPEG4 Encoder.
	XDM_LE_64	64-bit little endian stream. Not supported in this version of MPEG4 Encoder.
	XDM_BE_16	16-bit big endian stream. Not supported in this version of MPEG4 Encoder.
	XDM_BE_32	32-bit big endian stream. Not supported in this version of MPEG4 Encoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_BE_64	64-bit big endian stream. Not supported in this version of MPEG4 Encoder.
XDM_ChromaFormat	XDM_CHROMA_NA	Chroma format not applicable
	XDM_CHROMAFORMAT_DEFAULT	Default value is set to XDM_YUV_422ILE
	XDM_YUV_420P	YUV 4:2:0 planar
	XDM_YUV_422P	YUV 4:2:2 planar. Not supported in this version of MPEG4 Encoder.
	XDM_YUV_422IBE	YUV 4:2:2 interleaved (big endian)
	XDM_YUV_422ILE	YUV 4:2:2 interleaved (little endian)
	XDM_YUV_444P	YUV 4:4:4 planar. Not supported in this version of MPEG4 Encoder.
	XDM_YUV_411P	YUV 4:1:1 planar. Not supported in this version of MPEG4 Encoder.
	XDM_GRAY	Gray format. Not supported in this version of MPEG4 Encoder.
	XDM_RGB	RGB color format. Not supported in this version of MPEG4 Encoder.
XDM_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 Params 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.
	XDM_GETBUFINFO	Query algorithm instance regarding the properties of input and output buffers.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_GETVERSION	Query the algorithm's version. The result will be returned in the @c data field of the respective _Status structure.
	XDM_GETCONTEXTINFO	Query a split codec part for its context needs. Not supported in this version of MPEG4 Encoder.
XDM_EncodingPreset	XDM_DEFAULT	Default setting of the algorithm specific creation time parameters. This uses XDM_HIGH_QUALITY settings.
	XDM_HIGH_QUALITY	Set algorithm specific creation time parameters for high quality (default setting).
	XDM_HIGH_SPEED	Set algorithm specific creation time parameters for high speed. Not supported in this version of MPEG4 Encoder.
	XDM_USER_DEFINED	User defined configuration using advanced parameters. This uses XDM_HIGH_QUALITY settings.
XDM_EncMode	XDM_ENCODE_AU	Encode entire access unit. Default value.
	XDM_GENERATE_HEADER	Encode only header.
XDM_ErrorBit	XDM_PARAMSCHANGE	Bit 8 ❑ 1 - Sequence Parameters Change ❑ 0 - Ignore
	XDM_APPLIEDCONCEALME NT	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

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	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 0-7: Codec and implementation specific

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

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.

‖ **Fields**

Field	Data type	Input/ Output	Description
numBufs	XDAS_Int32	Input	Number of buffers
descs[XDM_MAX _IO_BUFFERS}	XDM1_Sungl eBufDesc	Input	Array of buffer descriptor

4.2.1.3 XDM_SingleBufDesc

‖ **Description**

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

‖ **Fields**

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

4.2.1.4 XDM1_SingleBufDesc

‖ **Description**

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

‖ **Fields**

Field	Data type	Input/ Output	Description
*buf	XDAS_Int8	Input	Pointer to the buffer
bufSize	XDAS_Int32	Input	Size of buffer in bytes
accessMask	XDAS_Int32	Output	If the buffer was not accessed by the algorithm processor (for example, it was filled by DMA or other hardware accelerator that does not write through the algorithm's 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 MPEG4 Encoder, the buffer details are:

❑ Number of input buffer required is 1 for YUV 422ILE and 3 for YUV 420P

❑ Number of output buffer required is 1

❑ The input buffer sizes (in bytes) for worst case PAL D1 (720 x 576) format are:

> For YUV 420P:
> Y buffer = 720 * 576
> U buffer = 360 * 288
> V buffer = 360 * 288
>
> For YUV 422ILE:
> Buffer = 720 * 576 * 2
>
> ❑ There is no restriction on output buffer size except that it should be enough to store one frame of encoded data.The output buffer size returned by the XDM_GETBUFINFO command assumes that the worst case output buffer size is (frameHeight*frameWidth)/2.
>
> ❑ When motion vector access is enabled, the size of the output buffer required to store the motion vector data is (8*number of MBs in a frame).

4.2.1.6   IVIDEO1_BufDesc

‖ **Description**

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

‖ **Fields**

Field	Data type	Input/ Output	Description
numBufs	XDAS_Int32	Input	Number of buffers
frameWidth	XDAS_Int32	Input	Width of the video frame
frameHeight	XDAS_Int32	Input	Height of the video frame
framePitch	XDAS_Int32	Input	Frame pitch use to store the frame
bufDesc[IVIDEO_MAX_YUV_BUFFERS]	XDM1_SingleBufDesc	Input	Pointer to the vector containing buffer addresses
extendedError	XDAS_Int32	Input	Extended error field. Not supported for recon bufs.
frameType	XDAS_Int32	Input	Indicates the decoded frame type as IVIDEO_FrameType enumerator type
topFieldFirstFlag	XDAS_Int32	Input	Flag to indicate when the application should display the top field first. Not supported in this version of MPEG4 Encoder.
repeatFirstFieldFlag	XDAS_Int32	Input	Flag to indicate when the first field should be repeated. Not supported in this version of MPEG4 Encoder.
frameStatus	XDAS_Int32	Input	Frame status of IVIDEO_Output. This field does not apply to encoder recon bufs.

Field	Data type	Input/ Output	Description
repeatFrame	XDAS_Int32	Input	Number of times the display process needs to repeat the display progressive frame. This field does not apply to encoder recon bufs.
contentType	XDAS_Int32	Input	Content type of the buffer
chromaFormat	XDAS_Int32	Input	XDM_Chroma buffer

4.2.1.7 IVIDEO1_BufDescIn

❙ **Description**

Buffer descriptor for input video buffers

❙ **Fields**

Field	Data type	Input/ Output	Description
numBufs	XDAS_Int32	Input	Number of buffers in bufDesc[ ]
frameWidth	XDAS_Int32	Input	Width of the video frame
frameHeight	XDAS_Int32	Input	Height of the video frame
framePitch	XDAS_Int32	Input	Frame pitch used to store the frame.
bufDesc[XDM_MAX_ IO_BUFFERS]	XDM1_Singl eBufDesc	Input	Picture buffers

4.2.1.8 IVIDENC1_Fxns

❙ **Description**

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

❙ **Fields**

Field	Data type	Input/ Output	Description
ialg	IALG_Fxns	Input	Structure containing pointers to all the XDAIS interface functions. For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

Field	Data type	Input/ Output	Description
*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.
encodingPreset	XDAS_Int32	Input	Encoding preset. Minimum Value = 0 Maximum Value =See `xdm_encodingpreset` enumeration for details. Default Value = XDM_DEFAULT
rateControlPreset	XDAS_Int32	Input	Rate control preset: See `IVIDEO_RateControlPreset` enumeration for details. 1,2,4,5 supported. Minimum Value = 1 Maximum Value = 5 Default Value = See `IVIDEO_RATECONTROLPRESET_DEFAULT`
maxHeight	XDAS_Int32	Input	Maximum video height to be supported in pixels. Minimum Value = 1 Maximum Value = 720 Default Value = 576
maxWidth	XDAS_Int32	Input	Maximum video width to be supported in pixels. Minimum Value = 1 Maximum Value = 1280 Default Value = 720
maxFrameRate	XDAS_Int32	Input	Maximum frame rate in fps * 1000 to be supported. Minimum Value = 1*1000 Maximum Value = 100*1000 Default Value = 30*1000

Field	Data type	Input/ Output	Description
maxBitRate	XDAS_Int32	Input	Maximum bit-rate to be supported in bits per second. Minimum Value = >0 Maximum Value = 20000000 Default Value = 4000000
dataEndianness	XDAS_Int32	Input	Endianness of input data. See XDM_DataFormat enumeration for details. Minimum Value = XDM_BYTE Maximum Value = XDM_BYTE 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. Not supported in this version of MPEG4 Encoder. Minimum Value = 1 Maximum Value = 1 Default Value = 1
inputChromaFormat	XDAS_Int32	Input	Input chroma format. See XDM_ChromaFormat enumeration for details. Values supported = 1, 3 , and 4 Minimum Value = 1 Maximum Value = 4 Default Value = XDM_CHROMAFORMAT_DEFAULT
inputContentType	XDAS_Int32	Input	Input content type. See IVIDEO_ContentType enumeration for details. Minimum Value = IVIDEO_PROGRESSIVE Maximum Value = IVIDEO_PROGRESSIVE Default Value = IVIDEO_CONTENTTYPE_DEFAULT
reconChromaFormat	XDAS_Int32	Input	Chroma formats for the reconstruction buffers. Default Value = XDM_CHROMA_NA Not supported in this version of MPEG4 Encoder.

Note:

The maximum video height and width supported are 576 and 720 pixels respectively.

The maximum bit rate supported is 20 mbps.

For the supported maxBitRate values, see Annex N in *ISO/IEC 14496-2*. The following fields of IVIDENC1_Params data structure are level dependent:

❑ maxHeight

❑ maxWidth

❑ `maxFrameRate`

❑ `maxBitRate`

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

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

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

❑ For `encodingPreset`, `XDM_HIGH_QUALITY` option is supported and all other options are mapped into `XDM_HIGH_QUALITY`.

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.
inputHeight	XDAS_Int32	Input	Height of input frame in pixels. Minimum Value = 1 Maximum Value = 720 Default Value = 576
inputWidth	XDAS_Int32	Input	Width of input frame in pixels. Minimum Value = 1 Maximum Value = 1280 Default Value = 720
refFrameRate	XDAS_Int32	Input	Reference or input frame rate in fps * 1000. For example, if the frame rate is 30, set this field to 30000. Minimum Value = 1* 1000 Maximum Value = 100 * 1000 Default Value = 30000
targetFrameRate	XDAS_Int32	Input	Target frame rate in fps * 1000. For example, if the frame rate is 30, set this field to 30000. Minimum Value = 1* 1000 Maximum Value = 100 * 1000 Default Value = 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. Minimum Value = 1* 1000 Maximum Value = 20000000 Default Value = 4000000

Field	Data type	Input/ Output	Description
intraFrameInterval	XDAS_Int32	Input	Interval between two consecutive intra frames. ❑ 0 - Only first I frame followed by all P frames ❑ 1 - No inter frames (all intra frames) ❑ 2 - Consecutive IP sequence (if no B frames) ❑ N - N-1 P sequences between I-frames. Minimum Value = 0 Default Value = 30
generateHeader	XDAS_Int32	Input	Encode entire access unit or only header. See XDM_EncMode enumeration for details. Minimum Value = 0 Maximum Value = 1 Default Value = XDM_ENCODE_AU
captureWidth	XDAS_Int32	Input	If the field is set to: ❑ 0 - Encoded image width is used as pitch. ❑ Greater than image width with even value only. capture width is used as pitch. Minimum Value = 0 Maximum Value = 3840 Default Value = 0 **Note**: Odd value is not supported.
forceFrame	XDAS_Int32	Input	Force current (immediate) frame to be encoded as specific frame type. See IVIDEO_FrameType for details. Minimum Value = IVIDEO_NA_FRAME Maximum Value = IVIDEO_I_FRAME Default Value = IVIDEO_FRAMETYPE_DEFAULT
interFrameInterval	XDAS_Int32	Input	Number of B frames between two reference frames. Default Value = 1 Not supported in this version of MPEG4 Encoder.
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 MPEG4 Encoder.. Default Value = 0

Note:

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

❑ inputHeight **<=** maxHeight

❑ inputWidth **<=** maxWidth

❑ refFrameRate **<=** maxFrameRate

❑ targetFrameRate **<=** maxFrameRate

The value of the refFrameRate and targetFrameRate should

> be the same.
>
> ❑ targetBitRate **<=** maxBitRate
>
> ❑ The inputHeight and inputWidth must be multiples of two.
>
> ❑ captureWidth >= imagewidth with even value is supported in this version of MPEG4 Encoder.
>
> ❑ VOL header is needed only at the start of sequence.VOL header cannot be inserted during run-time.
>
> ❑ Insertion of I -frame when run-time intra interval is changed.

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 same as this inputID field.
topFieldFirs tFlag	XDAS_Int32	Input	Flag to indicate the field order in interlaced content. . This field is only applicable for interlaced content (not progressive). Not supported in this version of MPEG4 Encoder.

4.2.1.12 IVIDENC1_Status

‖ **Description**

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

‖ **Fields**

Field	Data type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
extendedError	XDAS_Int32	Output	Extended error code. See XDM_ErrorBit enumeration for details.

Field	Data type	Input/ Output	Description
bufInfo	XDM_AlgBufInfo	Output	Input and output buffer information. See XDM_AlgBufInfo data structure for details. This field provides the application with the algorithm's buffer requirements.
data	XDM1_SingleBuf Desc	Input	Buffer descriptor for data passing.

4.2.1.13 IVIDENC1_OutArgs

‖ **Description**

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

‖ **Fields**

Field	Data type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
extendedError	XDAS_Int32	Output	Extended error code. See XDM_ErrorBit enumeration for details.
bytesGenerated	XDAS_Int32	Output	The number of bytes generated.
encodedFrameTy pe	XDAS_Int32	Output	Frame types for video. See IVIDEO_FrameType enumeration for details.
inputFrameSkip	XDAS_Int32	Output	Frame skipping modes for video. See IVIDEO_SkipMode enumeration for details.
reconBufs	IVIDEO_BufDesc	Output	Pointer to reconstruction buffer descriptor.
outputID	XDAS_Int32	Output	Output ID corresponding to the encoded buffer. This is also used to free up the corresponding image buffer for further use by client application code.
encodedBuf	XDM1_SingleBuf Desc	Output	The encoder fills the buffer with the encoded bit-stream. In case of sequences having I and P frames only, these values are identical to@c outBufs assed in IVIDENC1_Fxns::process().

> **Note**:
>
> The recon buffers are always padded by 2*UMV_PADDING pixels in horizontal and vertical dimensions. The UMV_PADDING value is 32.

4.2.2   MPEG4 Encoder Data Structures

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

❑   `IMP4VENC_Params`

❑   `IMP4VENC_DynamicParams`

❑   `IMP4VENC_InArgs`

❑   `IMP4VENC_Status`

❑   `IMP4VENC_OutArgs`

4.2.2.1   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, `IVIDENC1_Params`.

‖ Fields

Field	Data type	Input/ Output	Description
videnc_params	IVIDENC1_Params	Input	See IVIDENC1_Params data structure for details.
encodeMode	XDAS_Int32	Input	Encoding mode: ❑ 1 - MPEG4 mode . ❑ 0 - H.263 mode Minimum Value = 0 Maximum Value = 1 Default Value = 1
levelIdc	XDAS_Int32	Input	Profile level indication for MPEG4. ❑ Level 0-5 - Data is set to a minimum value of zero or a maximum value of 5 based on MPEG4 standards. This field has no significance for H.263 encode mode. Minimum Value = 0 Maximum Value = 5 Default Value =5
numFrames	XDAS_Int32	Input	Number of frames to be encoded. This parameter is used only for VM4 rate control inside encoder. Minimum Value = 1 Default Value =0x7fffffff

Field	Data type	Input/Output	Description
rcAlgo	XDAS_Int32	Input	Rate control method: ❑ 0 - Disable rate control ❑ 1 - TM5 (Not supported) ❑ 2 - TMN5 (Not supported) ❑ 3 - PLR1 ❑ 4 - PLR3 ❑ 5 - VM4 (Not supported) ❑ 6 - TM5_MOD (Not supported) ❑ 7 - Constrained VBR. This RC algorithm is used for low delay applications. ❑ 8 - PLR4. Minimum Value = 0 Maximum Value = 8 Default Value = 8 **Note**: TM5, TMN5, VM4 and, modified TM5 rate control methods are not supported in this version of MPEG4 Encoder.
vbvBufferSize	XDAS_Int32	Input	Size of VBV buffer for bit-stream (in multiples of 16 kbits) depending on profile and level. Minimum value to be set is 2. **Note**: VBV overflow/underflow may occur while encoding at 1 fps with I frame periodicity set to 1. Minimum Value = 2 Default Value = 112
useVOS	XDAS_Int32	Input	VOS header insertion: ❑ 1 - On. ❑ 0 - Off Minimum Value = 0 Maximum Value = 1 Default Value = 1
useGOV	XDAS_Int32	Input	GOV header insertion: ❑ 1 - On ❑ 0 - Off Minimum Value = 0 Maximum Value = 1 Default value is 0.
useDataPartition	XDAS_Int32	Input	Data partitioning: ❑ 1 - On ❑ 0 - Off Minimum Value = 0 Maximum Value = 1 Default Value = 0

Field	Data type	Input/ Output	Description
useRVLC	XDAS_Int32	Input	Reversible variable length code: ❑ 1 - On ❑ 0 - Off Minimum Value = 0 Maximum Value = 1 Default Value = 0
maxDelay	XDAS_Int32	Input	Maximum delay for rate control in terms of milliseconds.. Minimum Value = 100 Maximum Value = 30000 Default Value = 1000

Note:

❑ If the size field in IVIDENC1_Params structure is set to BASIC, then the default values are used for IMP4VENC_Params structure.

❑ The maxDelay field is applicable only for RateControlMethod = 7, and is ignored for other rate control methods. The delay value is in milliseconds and the recommended value for low delay applications is 300 ms.

❑ The following are the limitations on the maxDelay parameter of IMP4VENC_Params data structure:

> maxDelay >= 100

> maxDelay <= 30000

❑ RateControlMethod = 7 is tuned for low delay applications (e.g video tele conferencing), where periodic I-frames are not used (to maintain low bit-rate).

❑ RateControlMethod = 8 (PLR4) is a version of PLR3, which does not skip frames. This rate control is recommended for storage applicatons. This rate control is also chosen when RateControlPreset is set to IVIDEO_STORAGE.

❑ RateControlMethod = 4 (PLR3) is chosen when RateControlPreset is set to IVIDEO_LOW_DELAY.

❑ This version does not support VM4, TMN5, and modified TM5 rate control algorithms. The following algorithms are recommended instead:

> TMN5 - Use PLR3

> VM4 - Use PLR4

> Modified TM5 - Use TM5

❑ Due to restrictions put by MPEG4 standard on level 0, the following applies to simple profile level 0:

> If AC prediction is enabled, then rate control algorithm should not be used.That is, rcAlgo = 0 (constant QP).

> QPinter value has no impact when RC algo is disabled (rcAlgo
> = 0) and all frames encoded with QPIntra value. Qpinter =
> qpintra when rcalgo =0.

4.2.2.2   IMP4VENC_DynamicParams

‖ **Description**

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

‖ **Fields**

Field	Data type	Input/Output	Description
videnc_dynamicparams	IVIDENC1_DynamicParams	Input	See IVIDENC1_DynamicParams data structure for details.
resyncInterval	XDAS_Int32	Input	Insert resync marker (RM) after given specified number of bits. A value of zero implies do not insert. Minimum Value = 0 Maximum Value = 65535 Default Value = 0
hecInterval	XDAS_Int32	Input	Insert header extension code (HEC) after given specified number of packets. A value of zero implies do not insert. Minimum Value = 0 Maximum Value = 65536 Default Value = 0
airRate	XDAS_Int32	Input	Adaptive intra refresh in MBs per frame. This indicates the maximum number of MBs (per frame) that can be refreshed using AIR. Minimum Value = 0 Maximum Value = No:of MBs - 1 Default Value = 0
mirRate	XDAS_Int32	Input	Mandatory intra refresh rate for MPEG4 in MBs per frame. This indicates the number of MB's (per frame) that should be refreshed using MIR. Minimum Value = 0 Maximum Value = No:of MBs - 1 Default Value = 0

Field	Data type	Input/Output	Description
qpIntra	XDAS_Int32	Input	Default Quantization Parameter (QP) for I frame. Minimum Value = 1 Maximum Value = 31 Default Value = 8
qpInter	XDAS_Int32	Input	Quantization parameter for P frame. Minimum Value = 1 Maximum Value = 31 Default Value = 8
FCode	XDAS_Int32	Input	`f_code`. As in MPEG4 specifications, the maximum `MV length is 1 << f_code-1`. This field has no significance for H.263 encode mode. Minimum Value = 1 Maximum Value = 7 Default Value = 3
UseHpi	XDAS_Int32	Input	Half pixel interpolation: ❑ 1 - On (Must be on only) Minimum Value = 1 Maximum Value = 1 Default Value = 1
useAcPred	XDAS_Int32	Input	AC prediction enable flag: ❑ 1 - On ❑ 0 - Off Minimum Value = 0 Maximum Value = 1 Default Value = 0
lastFrame	XDAS_Int32	Input	Last frame flag ❑ 1 - Last frame ❑ 0 - Not last frame Minimum Value = 0 Maximum Value = 1 Default Value = 0
MVDataEnable	XDAS_Int32	Input	Flag to enable Motion vector access ❑ 1 - Enable ❑ 0 - Disable Minimum Value = 0 Maximum Value = 1 Default Value = 0

Field	Data type	Input/Output	Description
useUMV	XDAS_Int32	Input	Flag to enable/disable use of UMV (Unrestricted Motion Vector) ❑ 0 - Off ❑ 1 - On Minimum Value = 0 Maximum Value = 1 Default Value = 1

Note:

❑ If the size field in `IVIDENC1_DynamicParams` structure is set to `BASIC`, then the default values are used for `IMP4VENC_DynamicParams` structure.

❑ When `lastFrame = 1`, video sequence end code is inserted into the bit-stream.

❑ For H.263 (short header) mode, the minimum QP value allowed for `qpInter` and `qpIntra` is 4. Any QP value below 4 is clipped to 4. This is due to the 8-bit quantization used in H.263. It is applicable only for constant QP with rcAlgo = 0.

❑ For MPEG4, The minimum QP value used is 2 (any value below 2 is clipped to 2).

❑ The parameter ranges for some of the parameters listed above are as follows:

`resyncInterval` - Range is 0 to 65535 in bits. If the value is greater than the compressed frame size in bits, then it does not insert any resync marker. It is the number of bits after which resync marker is to be inserted. This decides the packet size.

`hecInterval` - Range is 0 to 65536 in packets. The number of packets is decided by how many resync markers are present. For example: a value of `hecInterval = 10` means insert HEC after 10 packets.

`airRate` - Range is 0 to number of MBs in a frame - 1. This indicates the maximum number of MB's to be refreshed as intra macro blocks in a P frame. For example, a value of `airRate` = 5 means that at most 5 MBs can be refreshed per frame through adaptive intra refresh.

`mirRate` - Range is 0 to number of MBs in a frame - 1. This indicates the number of MBs that must be refreshed as intra macro blocks in a P frame. For example, a value of `mirRate` = 5 means that 5 MBs must be refreshed per frame through intra refresh.

`vbvBufferSize` - The range of VBV buffer sizes for different resolutions is available in MPEG4 standard.

4.2.2.3   IMP4VENC_InArgs

‖ **Description**

> This structure defines the run-time input arguments for MPEG4 Encoder instance object. It uses the basic XDM data structure.

‖ **Fields**

Field	Data type	Input/Output	Description
videnc_InArgs	IVIDENC1_InArgs	Input	See IVIDENC1_InArgs data structure for details.

4.2.2.4   IMP4VENC_Status

‖ **Description**

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

‖ **Fields**

Field	Data type	Input/Output	Description
videnc_status	IVIDENC1_Status	Output	See IVIDENC1_Status data structure for details.
mvDataSize	XDAS_Int32	Output	Size of the motion data array in bytes.

4.2.2.5   IMP4VENC_OutArgs

‖ **Description**

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

‖ **Fields**

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

Note:

Use the field mvDataSize only when MVDataEnable parameter in IMP4VENC_DynamicParams is set to 1.

4.3 Interface Functions

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

❑ **Creation** – `algNumAlloc()`, `algAlloc()`

❑ **Initialization** – `algInit()`

❑ **Control** – `control()`

❑ **Data processing** – `algActivate()`, `process()`, `algDeactivate()`

❑ **Termination** – `algFree()`

You must call these APIs in the following sequence:

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

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

`algNumAlloc()`, `algAlloc()`, `algInit()`, `algActivate()`, `algDeactivate()`, and `algFree()` are standard XDAIS APIs. This document includes only a brief description for the standard XDAIS APIs. For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number 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's IALG functions. If an algorithm does not require a parent object to be created, this pointer must be set to NULL.

The third argument is a pointer to a memory space of size nbufs * sizeof(IALG_MemRec) where, nbufs is the number of buffers returned by algNumAlloc() and IALG_MemRec is the buffer-descriptor structure defined in ialg.h.

After calling this function, memTab[] is filled up with the memory requirements of an algorithm.

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

‖ **See Also**

algNumAlloc(), algFree()

4.3.2   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. See IMP4VENC_Params and IVIDENC1_Params for details. If this argument is set to NULL, API will take default values for all the parameters.

Note:

algInit() must assume default creation parameters and must not fail when params pointer is made NULL.

.

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

‖ Name

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

‖ Synopsis

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

‖ Arguments

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

`IVIDENC1_Cmd id; /* algorithm specific control commands*/`

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

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

‖ Return Value

`IALG_EOK; /* status indicating success */`

`IALG_EFAIL; /* status indicating failure */`

‖ Description

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

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

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

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

Note:

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

‖ Preconditions

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

❑ `control()` can only be called after a successful return from `algInit()` and `algActivate()`.

❑ If algorithm uses DMA resources, `control()` can only be called after a successful return from `DMAN3_init()`.

❑ `handle` must be a valid handle for the algorithm's instance object.

‖ Postconditions

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

❑ If the control operation is successful, the return value from this operation is equal to `IALG_EOK`; otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.

❑ If the control command is not recognized, the return value from this operation is not equal to `IALG_EOK`.

‖ Example

See test application file, TestAppEncoder.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's scratch buffers using the persistent memory that is part of the algorithm's instance object.

The first (and only) argument to algActivate() is an algorithm instance handle. This handle is used by the algorithm to identify various buffers that must be initialized prior to calling any of the algorithm's processing methods.

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

‖ See Also

　　　　　　　algDeactivate()

‖ **Name**

process() – basic encoding/decoding call

‖ **Synopsis**

```
XDAS_Int32 (*process)(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**

This function does the basic encoding/decoding. The first argument to process() is a handle to an algorithm instance.

The second and third arguments are pointers to the input and output buffer descriptor data structures respectively (see 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.

Note:

If you are using extended data structures, the fourth and fifth arguments must be pointers to the extended InArgs and OutArgs data structures respectively. Also, ensure that the size field is set to the size of the extended data structure. Depending on the value set for the size field, the algorithm uses either basic or extended parameters.

‖ **Preconditions**

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

❑ process() can only be called after a successful return from algInit() and algActivate().

❑ If algorithm uses DMA resources, `process()` can only be called after a successful return from `DMAN3_init()`.

❑ `handle` must be a valid handle for the algorithm's instance object.

❑ Buffer descriptor for input and output buffers must be valid.

❑ Input buffers must have valid input data.

‖ **Postconditions**

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

❑ If the process operation is successful, the return value from this operation is equal to `IALG_EOK`; otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.

❑ After successful return from `process()` function, `algDeactivate()` can be called.

‖ **Example**

See test application file, TestAppEncoder.c available in the \Client\Test\Src sub-directory.

‖ **See Also**

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

Note:

❑ A video encoder or decoder cannot be preempted by any other video encoder or decoder instance. That is, you cannot perform task switching while encode/decode of a particular frame is in progress. Pre-emption can happen only at frame boundaries and after `algDeactivate()` is called.

❑ The input data can be either in 8-bit YUV 4:2:0 or 8-bit 4:2:2 format. The encoder output is MPEG-4 encoded bit-stream.

❑ For more details on motion vector access API, see Appendix A.

‖ 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()

Motion Vector Access API

This appendix provides a detailed description of the Motion Vector Access API.

A.1 Description

Motion Vector Access API is part of the XDM `process()` call, used by the application 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. 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 signed SAD, as shown in the following table.

Motion Vector Horizontal Displacement (HD)	Signed 16 - bit integer
Motion Vector Vertical Displacement (VD)	Signed 16 - bit integer
SAD	Signed 32 - bit integer

The API returns the motion vector data in a single buffer with these three values interleaved in contiguous memory as shown in Figure A-1. *Motion Vector and SAD Buffer Organization.*.

| HD |
| VD | MB(0.0)
| SSE |

| HD |
| VD | MB (0.1)
| SSE |

| HD |
| VD | MB (0, num_mb_cols -1)
| SSE |

| HD |
| VD | MB (1.0)
| SSE |

| HD |
| VD | MB (num_mb_rows -1 , num_mb_cols -2)
| SSE |

| HD |
| VD | MB (num_mb_rows -1 , num_mb_cols -1)
| SSE |
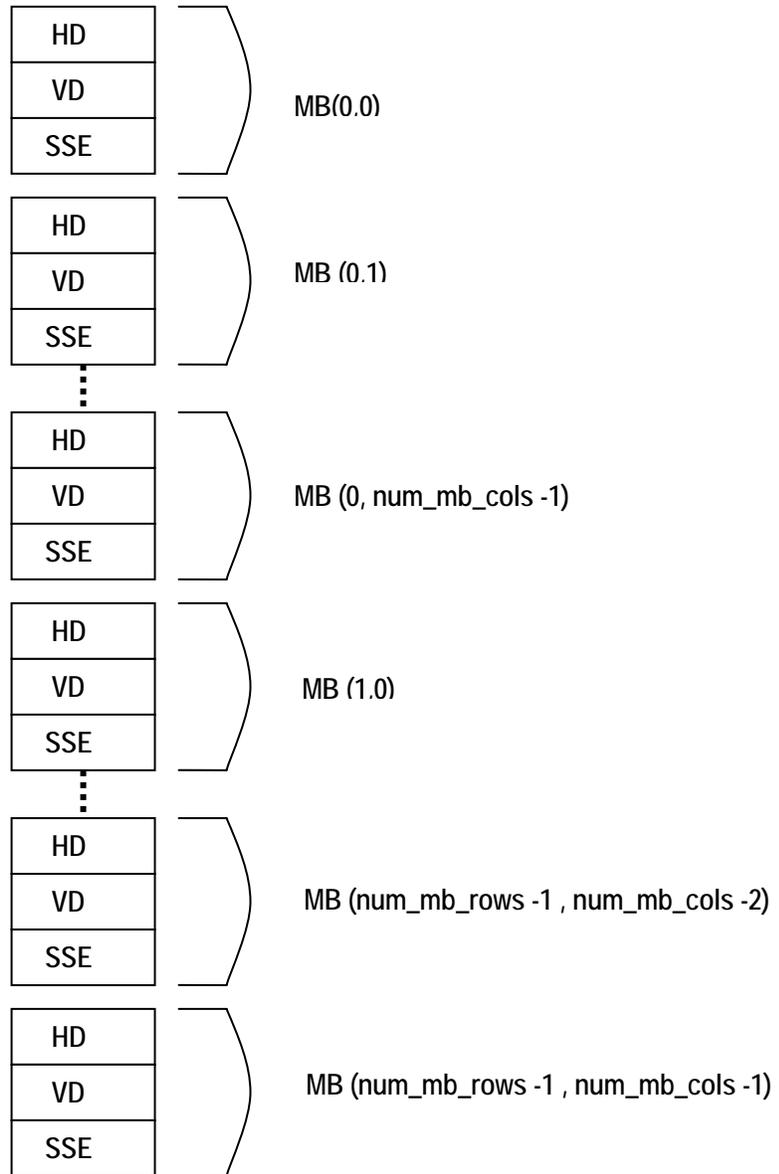
Figure A-1. Motion Vector and SAD Buffer Organization.

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 step1, this call will return the output buffer info as numBufs =2, along with the minimal buffer sizes.

   ```
   /* Initialize the output buffer descriptor */

   outputBufDesc.numBufs = 2;

   /* Stream Buffer */

   outputBufDesc.bufs[0]     =  streamDataPtr; //pointer
   to mpeg4 bit-stream

   outputBufDesc.bufSizes[0] =
   status.bufInfo.minOutBufSize[0];

   /* MV Buffer */

   outputBufDesc.bufs[1]     =  mvDtataPtr; //pointer to
   MV data

   outputBufDesc.bufSizes[1] =
   status.bufInfo.minOutBufSize[1];
   ```

3) Call 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 `status.mvDataSize`.

As shown in Figure A-1. *Motion Vector and SAD Buffer Organization.*, 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;
        int SAD;
} ;
motion_mbdata  *mbMV_data ;
mbMV_data =  (motion_mbdata  *)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 ->SAD;
mbMV_data ++;
     }
}
```

Note:

❑ The motion vectors are with half pel resolution. The half pel resolution MVs and ful pel MVs multiplied by 2 and added with the half pel MVs in the range [+/-1,+/-1].

❑ `SAD = abs(Ref(i,j) - Src(i,j))` where, `Ref` is the macro block of the reference region and `Src` is the macro block of the source image.

❑ The motion vectors seen in the encoded stream is based on the best coding decision, which is a combination of motion estimation and mode decission. The MV buffer returns the results of the motion estimation in full pel resolution (lowest SAD) and this maybe different from the motion vectors seen in the bit-stream. More details are given below :

   Some macro blocks in a P-frame may be coded as Intra macro

---

blocks based on the post motion estimation decisions. In this case, the motion vectors computed in the motion estimation stage (assuming that this macro block is inter) is returned.

Due to the post motion estimation decisions for some macro blocks, the actual motion vector encoded might be forced to (0,0). In this case, the non-zero motion vector that is available after the motion estimation is returned.

Some inter macroblocks may be not coded due to zero residual. In this case, the full pel motion vectors computed in the motion estimation stage is returned.

For I-frames, motion vectors are not returned and `status.mvDataSize = 0.`

This page is intentionally left blank

Revision History

This user guide revision history highlights the changes made to SPRUEA2B codec specific user guide to make it SPRUEA2C.

Table B-1.Revision History of MPEG4 Simple Profile Encoder on DM6446

Section	Additions/Modifications/Deletions
Global	❑ Modified Code Composer Studio version to 3.2.37.12 ❑ Modified DSP/BIOS version 5.32.02 ❑ Modified Framework Component (FC) version 2.20.00.15 ❑ Modified all instances of IVIDENC to IVIDENC1
Section 1.3	Supported Services and Features: ❑ Added Supports capture width ❑ Removed XDAIS compliant ❑ Updated XDM version to XDM 1.0 ❑ Added caution statement
Section 2.5.2	Encoder Configuration File: ❑ Modified Testparams.cfg
Section 4.1	Symbolic Constants and Enumerated Data Types: Added `IVIDEO_FrameType` Group or Enumeration Class and the following fields: ❑ `IVIDEO_NA_FRAME` ❑ `IVIDEO_I_FRAME` ❑ `IVIDEO_P_FRAME` ❑ `IVIDEO_B_FRAME` ❑ `IVIDEO_IDR_FRAME` ❑ `IVIDEO_II_FRAME` ❑ `IVIDEO_IP_FRAME` ❑ `IVIDEO_IB_FRAME` ❑ `IVIDEO_PI_FRAME` ❑ `IVIDEO_PP_FRAME` ❑ `IVIDEO_PB_FRAME` ❑ `IVIDEO_BI_FRAME` ❑ `IVIDEO_BP_FRAME` ❑ `IVIDEO_BB_FRAME` ❑ `IVIDEO_MBAFF_I_FRAME` ❑ `IVIDEO_MBAFF_P_FRAME` ❑ `IVIDEO_MBAFF_B_FRAME` ❑ `IVIDEO_MBAFF_IDR_FRAME` ❑ `IVIDEO_FRAMETYPE_DEFAULT` Removed the following Group or Enumeration Class: ❑ `IVIDEO_FrameType` ❑ `IVIDEO_ContentType` ❑ `IVIDEO_SkipMode`

Section	Additions/Modifications/Deletions
	Added `IVIDEO_ContentType` Group or Enumeration Class and the following fields: ❑ `IVIDEO_CONTENTTYPE_NA` ❑ `IVIDEO_PROGRESSIVE` ❑ `IVIDEO_PROGRESSIVE_FRAME` ❑ `IVIDEO_INTERLACED` ❑ `IVIDEO_INTERLACED_FRAME` ❑ `IVIDEO_INTERLACED_TOPFIELD` ❑ `IVIDEO_INTERLACED_BOTTOMFIELD` ❑ `IVIDEO_CONTENTTYPE_DEFAULT` Added following fields under `IVIDEO_RateControlPreset` Group or Enumeration Class: ❑ `IVIDEO_NONE` ❑ `IVIDEO_RATECONTROLPRESET_DEFAULT` Added `IVIDEO_SKIPMODE_DEFAULT` field under `IVIDEO_SkipMode` Group or Enumeration Class Added following fields under `XDM_DataFormat` Group or Enumeration Class: ❑ `XDM_BYTE` ❑ `XDM_LE_32` ❑ `XDM_LE_64` ❑ `XDM_BE_16` ❑ `XDM_BE_32` ❑ `XDM_BE_64` Added following fields under `XDM_ChromaFormat` Group or Enumeration Class: ❑ `XDM_CHROMA_NA` ❑ `XDM_CHROMAFORMAT_DEFAULT` Added following fields under `XDM_CmdId` Group or Enumeration Class Group or Enumeration Class ❑ `XDM_GETVERSION` ❑ `XDM_GETCONTEXTINFO` Added following fields under `XDM_ErrorBit` Group or Enumeration Class ❑ `XDM_PARAMSCHANGE`
Section 4.2	Added 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`

Section	Additions/Modifications/Deletions
	Removed the following Common XDM Data Structures: ❑ XDM_AlgBufInfo ❑ IVIDEO_BufDesc ❑ IVIDENC_Fxns ❑ IVIDENC_Params ❑ IVIDENC_DynamicParams ❑ IVIDENC_InArgs ❑ IVIDENC_Status ❑ IVIDENC_OutArgs
Section 4.2.1.10	Modified note under IVIDENC1_DynamicParams
Section 4.2.2	Modified maximum , minimum and default values for the following MPEG4 Encoder Data Structures: ❑ IMP4VENC_Params ❑ IMP4VENC_DynamicParams ❑ IMP4VENC_InArgs ❑ IMP4VENC_Status ❑ IMP4VENC_OutArgs
Section 4.2.2.1	IMP4VENC_Params: Added note under field RcAlgo.