MPEG4 Simple Profile Encoder on DM365

User's Guide

Literature Number: SPRUEV1B
February 2010

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

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

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

Read This First

About This Manual

This document describes how to install and work with Texas Instruments' (TI) MPEG4 Simple Profile Encoder implementation on the DM365 platform. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

TI's codec implementations are based on the eXpressDSP Digital Media (XDM) 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 DM365 platform.

This document assumes that you are fluent in the C language, have a good working knowledge of Digital Signal Processing (DSP), digital signal processors, and DSP applications. Good knowledge of eXpressDSP Algorithm Interface Standard (XDAIS) and eXpressDSP Digital Media (XDM), and IRES 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 – Revision History,** highlights the changes made to SPRUEV1A codec specific user guide to make it SPRUEV1B

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 API Reference (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 (SPRA579)* describes how to make algorithms compliant with the TMS320 DSP Algorithm Standard which is part of TI's eXpressDSP technology initiative.
- *xDAIS-DM (Digital Media) User Guide* (literature number SPRUEC8)
- *Using DMA with Framework Components for C64x+* (literature number SPRAAG1).

Related Documentation

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

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

Abbreviations

The following abbreviations are used in this document.

Table 1-1. List of Abbreviations.

Abbreviation	Description
API	Application Programming Interface
CBR	Constant Bit Rate
CSL	Chip Support Library
CVBR	Constrained Variable Bit Rate
DCT	Discrete Cosine Transform
DMA	Direct Memory Access
DMAN3	DMA Manager
EVM	Evaluation Module
GOB	Group of Blocks
GOV	Group of VOP

Abbreviation	Description
HEC	Header Extension Code
HPI	Half Pel Interpolation
IDMA3	DMA Resource specification and negotiation protocol
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ITU	International Telecommunications Union
MJCP	MPEG JPEG Co-Processor
MPEG	Moving Pictures Experts Group
QCIF	Quarter Common Intermediate Format
QP	Quantization Parameter
QVGA	Quarter Video Graphics Array
SP	Simple Profile
VBR	Variable Bit Rate
VBV	Video rate Buffer Verifier
VICP	Video and Imaging Co-Processor
VOL	Video Object Layer
VOP	Video Object Plane
VOS	Video Object Sequence
UMV	Unrestricted Motion Vector
XDAIS	eXpressDSP Algorithm Interface Standard
XDM	eXpressDSP Digital Media

Note:

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

Text Conventions

The following conventions are used in this document:

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

Product Support

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

Trademarks

Code Composer Studio and eXpressDSP are trademarks of Texas Instruments.

All trademarks are the property of their respective owners.

Contents

Figures

This page is intentionally left blank

Tables

This page is intentionally left blank

Chapter 1

Introduction

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

Topic **Page**

1.1 Overview of XDAIS, XDM, and IRES

TI's multimedia codec implementations are based on the eXpressDSP Digital Media (XDM) standard. XDM is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS). IRES is the interface for management and utilization of special resource types such as hardware accelerators, certain types of memory and DMA. This interface allows the client application to query and provide the algorithm its requested resources.

1.1.1 XDAIS Overview

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

- `algAlloc()`
- `algInit()`
- `algActivate()`
- `algDeactivate()`
- `algFree()`

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

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

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

1.1.2 XDM Overview

In the multimedia application space, you have the choice of integrating any codec into your multimedia system. For example, if you are building a video encoder system, you can use any of the available video encoders (such as MPEG4, H.263, or H.264) in your system. To enable easy integration with the client application, it is important that all codecs with

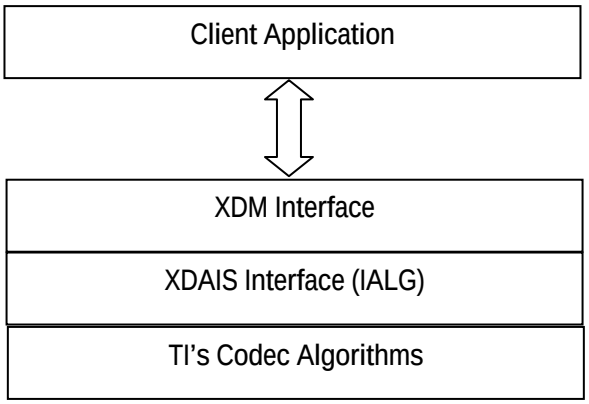
similar functionality use similar APIs. XDM was primarily defined as an extension to XDAIS to ensure uniformity across different classes of codecs (for example, audio, video, image, and speech). The XDM standard defines the following two APIs:

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

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

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

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



As shown 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 encoder, then you can easily replace MPEG4 with another XDM-compliant video encoder with minimal changes to the client application.

For more details, see *xDAIS-DM (Digital Media) User Guide* (SPRUEC8b).

1.1.3 IRES Overview

IRES is a generic, resource-agnostic, extendible resource query, initialization and activation interface. The application framework defines, implements, and supports concrete resource interfaces in the form of IRES extensions. Each algorithm implements the generic IRES interface, to request one or more concrete IRES resources. IRES defines standard interface functions that the framework uses to query, initialize, activate/deactivate and reallocate concrete IRES resources. To create an

algorithm instance within an application framework, the algorithm and the application framework agrees on the concrete IRES resource types that are requested. The framework calls the IRES interface functions, in addition to the IALG functions, to perform IRES resource initialization, activation, and deactivation.

The IRES interface introduces support for a new standard protocol for cooperative preemption, in addition to the IALG-style non-cooperative sharing of scratch resources. Co-operative preemption allows activated algorithms to yield to higher priority tasks sharing common scratch resources. Framework components include the following modules and interfaces to support algorithms requesting IRES-based resources:

1) **IRES** - Standard interface allowing the client application to query and provide the algorithm with its requested IRES resources.

2) **RMAN** - Generic IRES-based resource manager, which manages and grants concrete IRES resources to algorithms and applications. RMAN uses a new standard interface, the IRESMAN, to support run-time registration of concrete IRES resource managers.

Client applications call the algorithm's IRES interface functions to query its concrete IRES resource requirements. If the requested IRES resource type matches a concrete IRES resource interface supported by the application framework, and if the resource is available, the client grants the algorithm logical IRES resource handles representing the allotted resources. Each handle provides the algorithm with access to the resource as defined by the concrete IRES resource interface.

IRES interface definition and function calling sequence is depicted in the following figure. For more details, see *Using IRES and RMAN Framework Components for C64x+* (literature number SPRAAI5).

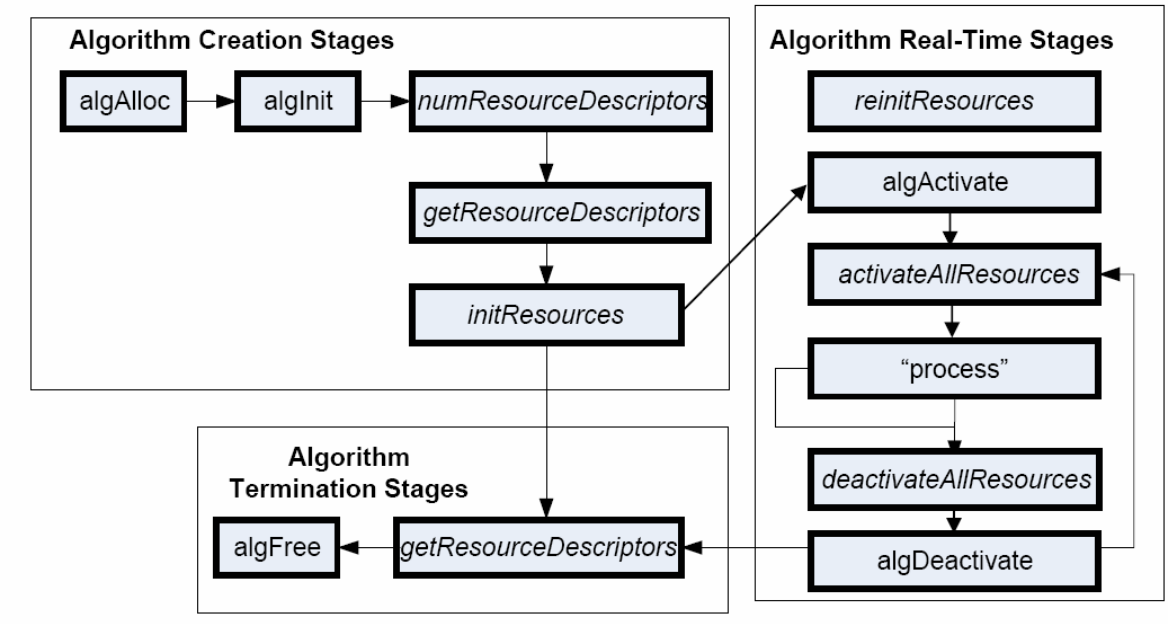


Figure 1-1. IRES Interface Definition and Function Calling Sequence.

For more details, see *Using IRES and RMAN Framework Components for C64x+* (literature number SPRAAI5).

1.2 Overview of MPEG4 Simple Profile Encoder

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

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

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 DM365 platform.

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

- eXpressDSP Digital Media (XDM 1.0 IVIDENC1) interface and IRES compliant
- Compliant with the MPEG4 simple profile levels 0, 1, 2, 3. In addition, it can encode 720P (1280x720),SXVGA (1280x960) and 1080P (1920x1080) formats.
- Supports YUV 4:2:2 interleaved data as an input
- Supports YUV 4:2:0 semi-planar (NV12 format, that is, Y planar Cb Cr interleaved) data as an input
- Supports image width as multiple of 16 and height as multiple of 16
- Supports Half Pel Interpolation (HPI) for motion estimation
- Supports one motion vector encoding for motion estimation (1MV/MB) with (-32, +31) half pel search range
- Supports 21H (low quality, high performance) or 8421H (medium quality, medium performance) or 44421H (high quality, medium performance) or 84221H (High quality, low performance) based on meAlgo API parameter.
- Supports DC prediction
- Supports AC prediction when rate control is disabled, that is, fixed Qp mode
- Supports generation of streams with Resync Marker (RM)
- Supports MPEG2 Step 2 TM5 rate control algorithm
- Supports Variable Bit Rate (`IVIDEO_STORAGE`), Constant Bit Rate (`IVIDEO_LOW_DELAY`), Fixed Qp (`IVIDEO_NONE`) and Constrained Variable BitRate (CVBR) (see section 4.2.2)

- Supports Intra – Inter decision at 16x16 level (for better speed) or 8x8 block level (for better quality) level based on intraAlgo API parameter
- Supports Bonus Skip MB logic (for better quality) or non-Bonus Skip MB logic (for better performance) based on `SkipMBAlgo` API parameter
- Supports Unrestricted Motion Vectors (UMV)
- Supports access of motion vectors and SAD through MV access API. The application should pass the buffer required to write the SAD and motion vector generated. This should be passed as an output buffer parameter. MV access API always provides the motion vectors for the best matching MB
- Supports the VOL header generation at frame-level. The application has to pass the buffer required to write the VOL header generated. The encoding process is by-passed and frame count is unaltered when the Header generation API is called.
- Supports modification of target bit-rate and frame rate
- Supports setting of separate Quantization Parameter (Qp) for I-frames and P-frames
- Supports changing the size of video packets at create time
- Supports area encode. The application can provide width, height, sub window width, and sub window height to the algorithm for encoding. The sub-window width and sub-window height should be multiple of 16.
- Supports rotation (90, 180 and 270 degrees) integrated with the Encoder up to a resolution of 720x576.
- Supports changing the encoding parameters at run-time (dynamic configurability)
- Supports frame level reentrancy
- Supports multi-instance of MPEG4 Encoder and single/multi instance of MPEG4 Encoder with other DM365 codecs
- Supports insertion of user data by application (see section 3.6)

1.4 Limitations

This encoder does not support the following:

- Does not support 4 MV
- Does not support AC prediction for varying Qp
- Does not support ME range beyond -32 and +31. Only ME Range = 31 and ME Range = 7 are supported
- Does not support DP, RVLC and HEC

- Does not support input width/height, sub-window width/height, rate control algorithm, VBV size, or rotation as dynamically configurable parameters
- Does not support arbitrary width and height
  - Supports image width as multiple of 16 and height as multiple of 8
  - Does not support image width below 160 (without UMV)
  - Does not support image width below 192 (with UMV)
- Does not support rotation with width more than 720 or height more than 576 (for instance, 720p (1280x720) or SXVGA (1280x960))
- Does not support area encode feature with horizontal and vertical offsets
- Does not support image width more than 1920 and image height more than 1920

This page is intentionally left blank

Chapter 2

Installation Overview

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

Topic **Page**

2.1 System Requirements for NO-OS Standalone

This section describes the hardware and software requirements for the normal functioning of the codec component in Code Composer Studio. For details about the version of the tools and software, see Release Note.

2.1.1 Hardware

- DM365 EVM (Set the bits 2 and 3 of switch SW4 to high(1) position; Set the bits 4 and 5 of SW5 to high(1) position)
- XDS560R JTAG

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 3.3.81.6 (Service Release-11)
- **Code Generation Tools:** This project is compiled, assembled, archived, and linked using the TI ARM code generation tools
- DM365 functional simulator

2.2 System Requirements for Linux

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

2.2.1 Hardware

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

2.2.2 Software

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

- **Linux:** Monta Vista Linux 5.0
- **Code Generation Tools:** This project is compiled, assembled, and linked using the arm_v5t_le-gcc compiler.

2.3 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 210_V_MPEG4_E_01_10, under which another directory named mpeg4_encoder is created.

Figure 2-1 shows the sub-directories created in the mpeg4_encoder directory.

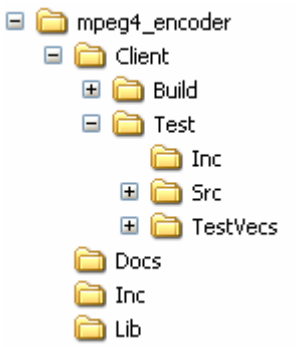


Figure 2-1. Component Directory Structure.

Table 2-1 provides a description of the sub-directories created in the mpeg4_encoder directory.

Table 2-1. Component Directories.

Sub-Directory	Description
mpeg4_encoder/Client/Build	This folder is available only in NO-OS Standalone release package. Not required for Linux release package. Contains make file, cmd file and configuration file to build the NO-OS standalone test application.
mpeg4_encoder/Docs	Contains user guide, datasheet, and release notes
mpeg4_encoder/Client/Test/Src	Contains test application C files, makefile, and configuration file. Executable will be built in this folder.
mpeg4_encoder/Client/Test/Inc	Contains header files needed for the application code
mpeg4_encoder/Client/Test/TestVecs	Contains test vectors, configuration files
mpeg4_encoder/Inc	Contains the interface file for MPEG4 Encoder
mpeg4_encoder/Lib	Contains MPEG Encoder and other support libraries

2.4 Building the Sample Test Application for EVM Standalone (NO-OS)

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

1) Verify that you have an installation of TI's Code Composer Studio version 3.3.81.6 (Service Release-11) and code generation tools as provided in the Release Note.

2) Verify if the codec object library mp4vencAlg.lib exists in the \Lib sub-directory.

3) Ensure that you have installed the XDC and Framework components releases with version numbers that are mentioned in the release notes.

4) For installing framework component, unzip the content at some location and set the path of the base folder in FC_INSTALL_DIR environment variable

5) Ensure that the installed XDC directory is in the general search PATH.

6) Open the MS-DOS command prompt at the directory \Client\Build\ sub-directory of the release folder.

7) Type the command "**gmake –f mp4vencTestApp.mak**" at the prompt and this generates an executable file, mp4vencApp.out in the \Client\Build\Out sub-directory.

2.5 Running the Sample Test Application on EVM Standalone (NO-OS)

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

1) Verify that you have an installation of TI's Code Composer Studio version 3.3.81.6 with Service Release 11 and code generation tools as provided in the Release Note.

2) Verify the SDXDS560R JTAG driver installation version 30329A.

3) Check SW4 and SW5 switch positions of the DM365 EVM. Bit 2 and 3 of SW4 should be set to 1 and remaining should be set to 0. All bits should be set to 0 for SW5.

4) Open Setup Code Composer Studio version 3.3.

5) Select **File > Import**, browse for the .ccs file, and add it.

6) Save the configuration and exit from setup Code Composer Studio. **PDM** opens and displays both ARM926 and ARM968 processors.

7) Right click on ARM926 and connect.

8) Double-click ARM926 to launch Code Composer Studio IDE for the host processor.

9) Add the GEL file and initialize it properly

10) Select **File > Load Program** in Host Code Composer Studio, browse to the \Client\Build\Out\ sub-directory, select the codec executable created in Section 2.4, and load it into Code Composer Studio in preparation for execution.

11) Select **Debug > Run** in Host Code Composer Studio to execute encoder on host side.

    The sample test application takes the input files stored in the \Client\Test\Testvecs\Input sub-directory, runs the codec, and stores the output in \Client\Test\Testvecs\Output sub-directory.

    For each encoded frame, the application displays a message indicating the frame number and the bytes generated.

After the encoding is complete, the application displays a summary of total number of frames encoded.

12) Halt the coprocessor from Code Composer Studio IDE.

2.6 Building and Running the Sample Test Application on Linux

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

1) Verify that you have installed Framework Component (FC), XDC, and LSP. For information about the version, see Release Note.

2) Verify that libmp4venc.a library is present in mpeg4_encoder/Lib directory.

3) Change directory to mpeg4_encoder/Client/Test/Src and type `make clean` followed by a `make` command. This will use the makefile in that directory to build the test executable mp4enc into the mpeg4_encoder/Client/Test/Src directory.

Note:

The ARM tool chain, arm_v5t_le-gcc (ARM gcc), compiler path needs to be set in your environment path before building the MPEG4 encoder executable.

To run mp4enc executable on DM365 EVM board, follow these steps:

4) Set up the DM365 EVM Board. For information about setting up the DM365 environment, see the *DM365 Getting Started Guide* available in the doc directory in DVSDK release package.

5) Run the MPEG4 Encoder executable:

   a) Ensure that complete Client folder is in target file system

   b) Copy the kernel modules cmemk.ko, edmak.ko and irqk.ko to the target directory. These modules are provided with the release package in kernel_modules directory.

   c) Copy loadmodules.sh provided with release package at kernel_modules to the target directory.

   d) Load the kernel modules by executing folliwng command.

      ```
      $./loadmodules.sh
      ```

      Change the directory to Client/Test/Src folder and execute following command to run the MPEG4 encoder executable

      ```
      $./mp4enc-r
      ```

   This will run the MPEG4 Encoder with base parameters.

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

```
$./mp4enc-r –ext
```

2.7 Configuration Files

This codec is shipped along with:

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

2.7.1 Generic Configuration File

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

The format of the Testvecs.cfg file is:

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

where:

- `X` may be set as:
  - 0 - for output dumping
- `Config` is the Encoder configuration file. For details, see Section 2.7.2.
- `Input` is the input file name (use complete path).
- `Output` is the output file name.

A sample Testvecs.cfg file is as shown.

```
0
../TestVecs/Config/Testparams.cfg
../TestVecs/Input/colorful_toys_cif_5frms_420SP.yuv
../TestVecs/Output/colorful_toys_cif_5frms.bits
```

2.7.2 Encoder Configuration File

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

A sample Testparams.cfg file is as shown.

```
# New Input File Format is as follows
# <ParameterName> = <ParameterValue> # Comment
```

```
#
##############################################################################
# Parameters
##############################################################################
ImageHeight      = 288     # Image height in Pels, must be
                             multiple of 16
ImageWidth       = 352     # Image width in Pels, must be
                             multiple of 16
FrameRate        = 30000   # Frame Rate per second*1000
Bitrate          = 737280  # Bitrate(bps)
                             This parameter is ignored for
                             IVIDEO_NONE
ChromaFormat     = 4       # 4 =>
                             XDM_YUV_422ILE, 9=> YUV_420 SemiPlanar
                             Others not supported
RCAlgo               = 0   # 0: VBR 1: CBR  2: fixedQp
subWindowHeight  = 288     # Height of the Subwindow, must be
                             multiple of 16
subWindowWidth   = 352     # Width of the Subwindow, must be
                             multiple of 16
IntraPeriod      = 30      # Period of I-Frames (N-1) P frames
                             (Non-negative)
intraAlgo        = 1       # INTRA/INTER Decision Algorithm.
                             0: IMP4VENC_INTRA_INTER_DECISION_LQ_HP
                             1: IMP4VENC_INTRA_INTER_DECISION_HQ_LP
numMBRows        = 5       # Number of MB rows in a Packet.
                             Maximum value is "subWindowHeight/16"
initQ            = 0       # Initial Q (at picture head).
                             0:automatically determined,
                             1-31:force initial Q
rcQ_MAX          = 31      # Q MAX value  (1-31)
rcQ_MIN          = 2       # Q MIN value  (1-31)
rateFix          = 0       # Reserved
rateFixRange     = 0       # Reserved
rotation         = 0       # Rotation (anticlockwise)
                             0: No Rotation,
                             90: 90 degree,
                             180: 180 degree,
                             270: 270 degree
meAlgo           = 1       # Motion estimation algorithm
                             0: IMP4VENC_ME_MQ_MP
                             1: IMP4VENC_ME_HQ_MP
                             2: IMP4VENC_ME_HQ_LP
                             3: IMP4VENC_ME_LQ_HP
SkipMBAlgo       = 0       # 0: IMP4VENC_SKIP_MB_LQ_HP
                             1: IMP4VENC_SKIP_MB_HQ_LP
UMV              = 0       # 0: IMP4VENC_UMV_LQ_HP
                             1: IMP4VENC_UMV_HQ_LP
VBV_size         = 100     # Video rate Buffer verifier size in 16 kb.
                           Number depending on the resolution of
                           video frame, typically equal to (No. of MB's
                           per frame x110 x FrameRate) /(16000 * 1000)
```

Any field in the `IVIDENC1_Params` structure (see Section 4.2.1.6) can be set in the Testparams.cfg file using the syntax shown previously. If you specify additional fields in the Testparams.cfg file, ensure to modify the test application appropriately to handle these fields.

- If `initQ` is not specified through an extended parameter, the default value is calculated by codec. `VBV_size` should be of the order of (No. of MBs per framex110x FrameRate) /(16000 * 1000) for better quality.
- The best quality is achieved with the following parameter settings:

```
{
intraAlgo  = IMP4VENC_INTRA_INTER_DECISION_HQ_LP
MeAlgo     = IMP4VENC_ME_HQ_LP
SkipMBAlg0 = IMP4VENC_SKIP_MB_HQ_LP
UMV        = IMP4VENC_UMV_HQ_LP
}
```

- The best performance is achieved with the following parameter settings:

```
{
intraAlgo    = IMP4VENC_INTRA_INTER_DECISION_LQ_HP
MeAlgo = IMP4VENC_ME_LQ_HP
SkipMBAlg0   = IMP4VENC_SKIP_MB_LQ_HP
UMV          = IMP4VENC_UMV_LQ_HP

}
```

- Typical bit-rates for different resolutions are as follows:

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

2.7.3 Encoder Sample Base Param Setting

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

```
typedef struct IVIDENC1_Params {
XDAS_Int32 size;
XDAS_Int32 encodingPreset = XDM_HIGH_SPEED; // Value = 2
XDAS_Int32 rateControlPreset = IVIDEO_STORAGE; //value = 2
XDAS_Int32 maxHeight = 720;
XDAS_Int32 maxWidth = 1280;
XDAS_Int32 maxFrameRate = 30000;
XDAS_Int32 maxBitRate = 10000000;
XDAS_Int32 dataEndianness = XDM_BYTE;
XDAS_Int32 maxInterFrameInterval = 1;
XDAS_Int32 inputChromaFormat = XDM_YUV_420SP; //value = 9
XDAS_Int32 inputContentType = IVIDEO_PROGRESSIVE;
XDAS_Int32 reconChromaFormat XDM_YUV_420SP; //value = 9;
} IVIDENC1_Params;

typedef struct IVIDENC1_DynamicParams {
XDAS_Int32 size; /**< @sizeField */
XDAS_Int32 inputHeight; /**< Input frame height. */
XDAS_Int32 inputWidth; /**< Input frame width. */
XDAS_Int32 refFrameRate = 30000;
XDAS_Int32 targetFrameRate = 30000;
XDAS_Int32 targetBitRate; < 10000000 /**< Target bit rate in bits per second. */
XDAS_Int32 intraFrameInterval = 30;
XDAS_Int32 generateHeader = 0;
XDAS_Int32 captureWidth; // for demo, same as inputWith
XDAS_Int32 forceFrame; = IVIDEO_NA_FRAME
XDAS_Int32 interFrameInterval = 0;
XDAS_Int32 mbDataFlag = 0;
} IVIDENC1_DynamicParams;

typedef struct IVIDENC1_InArgs {
XDAS_Int32 size; /**< @sizeField */
XDAS_Int32 inputID; /* as per application*/
XDAS_Int32 topFieldFirstFlag = 0;
} IVIDENC1_InArgs;
```

2.7.4 Encoder Sample Extended Param Setting

The encoder can be run in `IMP4VENC extended` class setting. The base parameter variables of encoder can be set as mentioned in preceding section. The following list provides the typical values of `IMP4VENC extended` class variables.

```
typedef struct IMP4VENC_Params {
IVIDENC1_Params videncParams; /* Set the Base parameter structure */
XDAS_Int32 subWindowHeight = <Input Height>; /* Height of the Subwindow */
XDAS_Int32 subWindowWidth = <Input Width>;  /* Width of the Subwindow */
XDAS_Int32 rotation = 0;
XDAS_Int32 vbvSize = 0;
XDAS_Int32 svhMode =0;
XDAS_Int32 IFrameBitRateBiasFactor = 0;
XDAS_Int32 PFrameBitRateBiasFactor = 0;
XDAS_Int32 peakBufWindow = 0;
XDAS_Int32 minBitRate = <bitRate>;
} IMP4VENC_Params;

typedef struct IMP4VENC_DynamicParams
{
IVIDENC1_DynamicParams videncDynamicParams; /*Set the Base Dynamic parameter
                                             structure */
```

```
XDAS_Int32 intraAlgo = 0;
XDAS_Int32 numMBRows = 0;
XDAS_Int32 initQ = 0;
XDAS_Int32 rcQMax = 31;
XDAS_Int32 rcQMin = 1;
XDAS_Int32 intraFrameQP = 0;
XDAS_Int32 interFrameQP = 0;
XDAS_Int32 rateFix = 0;
XDAS_Int32 rateFixRange = 0;
XDAS_Int32 meAlgo = 0;
XDAS_Int32 skipMBAlgo = 0;
XDAS_Int32 unrestrictedMV = 0;
XDAS_Int32 mvDataEnable = 0;
}IMP4VENC_DynamicParams;

typedef struct IMP4VENC_InArgs
{
IVIDENC1_InArgs videncInArgs; /* Set the Base InArgs structure */
XDAS_Int32 subWindowHozOfst = 0;
XDAS_Int32 subWindowVerOfst = 0;
XDAS_Bool insertUserData = 0;
XDAS_UInt32 lengthUserData = 0;
} IMP4VENC_InArgs;
```

2.8 Uninstalling the Component

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

Chapter 3

Sample Usage

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

Topic **Page**

3.1 MPEG4 Encoder Client Interfacing Constraints

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

1) DMA requirements of MPEG4 Encoder: Current implementation of the MPEG4 encoder uses the following number of TCCs and PaRamSets for its DMA resource requirements.

TCC Requirement	PaRamSet Requirement
28 TCCs (TCC #63 compulsory)	All paramsets associated with TCCs and 38 additional paramsets.

2) Channel mapping to queue and EDMA shadow region setting is done by codec.

3) If there are multiple instances of a codec and/or different codec combinations, the application can use the same group of channels and PaRAM entries across multiple codecs. The `AlgActivate` and `AlgDeactivate` calls made by client application and implemented by the codecs, perform context save/restore to allow multiple instances of same codec and/or different codec combinations.

4) As all codecs use the same hardware resources, only one process call per codec should be invoked at a time (frame level reentrancy). The process call needs to be wrapped within activate and deactivate calls for context switch. See XDM specification on activate/deactivate.

5) If multiple codecs are running with frame level reentrancy, the client application has to perform time multiplexing of process calls of different codecs to meet the desired timing requirements between video/image frames.

6) The ARM and DDR clock must be set to the required rate for running single or multiple codecs.

7) The codec combinations feasibility is limited by processing time (computational hardware cycles) and DDR bandwidth.

8) Codec atomicity is supported at frame level processing only. The process call has to run until completion before another process call can be invoked.

3.2 Overview of the Test Application

The test application exercises the IMP4VENC_Params extended class of the MPEG4 Encoder library.. The main test application files are mpeg4EncTestApp.c and TestAppEncoder.h. These files are available in the /Client/Test/Src and /Client/Test/Inc sub-directories, respectively.

The following figure depicts the sequence of APIs exercised in the sample test application.

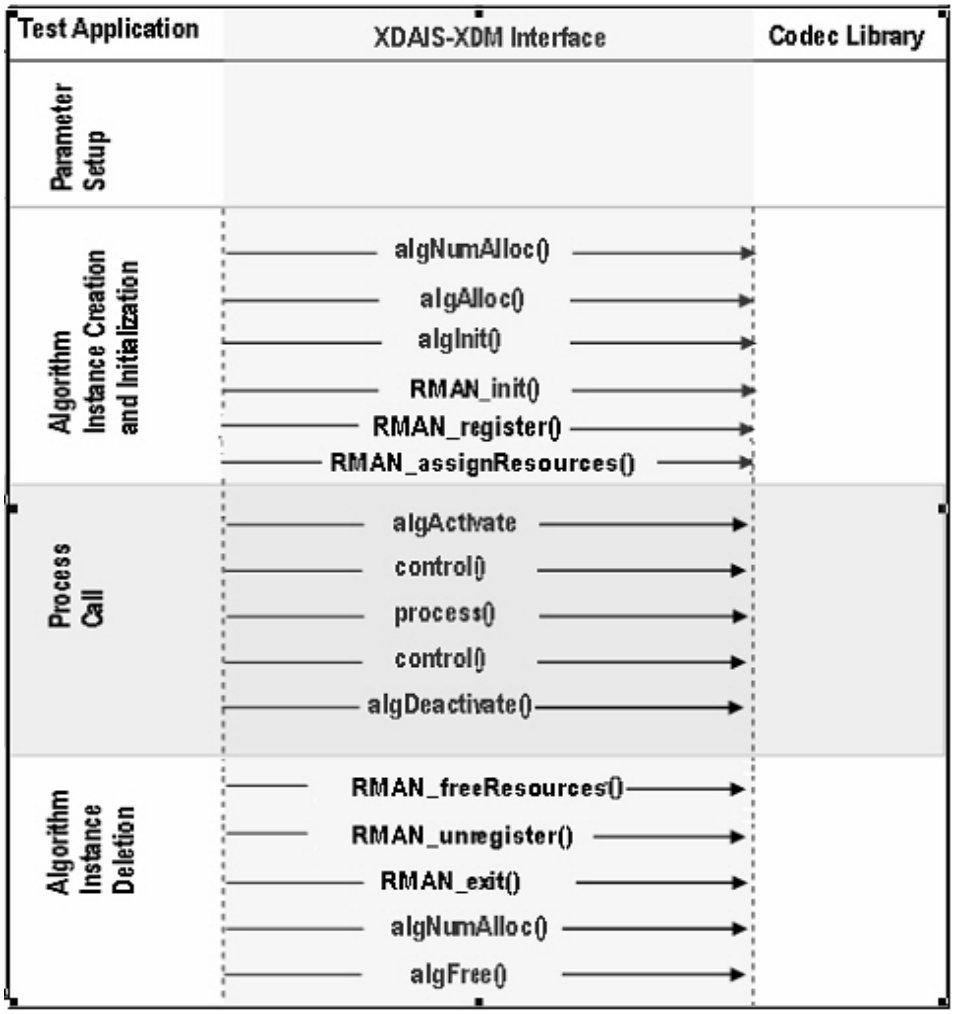


Figure 3-1. Test Application Sample Implementation.

The test application is divided into four logical blocks:

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

3.2.1 Parameter Setup

Each codec component requires various codec configuration parameters to be set at initialization. For example, a video codec requires parameters such as video height, video width, 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 2.7.

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

4) Reads the input bit-stream into the application input buffer.

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

3.2.2 Algorithm Instance Creation and Initialization

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

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

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

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

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

1) `RMAN_init`: To initialize the RMAN module.

2) `RMAN_register`: To register the VICP protocol/resource manager with generic resource manager.

3) `RMAN_assignresources`: To register resources to the algorithm as requested VICP protocol/resource manager.

3.2.3 Process Call in Single Instance Scenario

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

1) Calls `algActivate()`, which initializes the encoder state and some hardware memories and registers.

2) Sets the input and output buffer descriptors required for the `process()` function call.

3) Calls the `process()` function to encode a single frame of data. The inputs to the process function are input and output buffer descriptors, the pointer to the `IVIDENC1_InArgs` and `IVIDENC1_OutArgs` structures. `process()` function should be called multiple times to encode multiple frames.

4) Call `algDeactivate()`, which performs releasing of hardware resources and saving of encoder instance values.

5) `process()` is made a blocking call, but an internal OS specific layer enables the process to be pending on a semaphore while hardware performs a complete MPEG4 encode.

6) Other specific details of the `process()` function remain the same.

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

3.2.4 Algorithm Instance Deletion

After successful execution of algorithm the test application frees up the DMA and VICP resource allocated for algorithm. This is implemented by calling RMAN interface functions in following sequence:

1) `RMAN_freeResources( )`: To free the resources allocated to the algorithm before process call.

2) `RMAN_unregister( )`: To unregister VICP protocol/resource manager with the generic resource manager.

3) `RMAN_exit( )`: To delete the generic IRES RMAN and release the memory.

After this, the test application must delete the current algorithm instance. The following APIs are called in sequence:

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

2) `algFree()` - To query the algorithm to get the memory record information and then free them up for the application

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

3.3 Usage in Multiple Instance Scenario

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

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

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

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

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

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

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

3.3.1 Process Call with algActivate and algDeactivate

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

1) Sets the input and output buffer descriptors required for the `process()` function call.
2) Calls `algActivate()`, which initializes the encoder state and some hardware memories and registers.
3) Calls the `process()` function to encode a single frame of data. The inputs to the process function are input and output buffer descriptors, and the pointer to the `IVIDENC1_InArgs` and `IVIDENC1_OutArgs` structures.

4) Calls `algDeactivate()`, which releases hardware resources and saves the decoder instance values.

5) Other specific details of the process() function remain the same.

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

3.4 Usage for Motion Vector Access

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

3.4.1 Description

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

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

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

> **Note**:
>
> The current version of the MPEG4 encoder stores the SAD (Sum of Absolute Differences) in place of SSE.

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

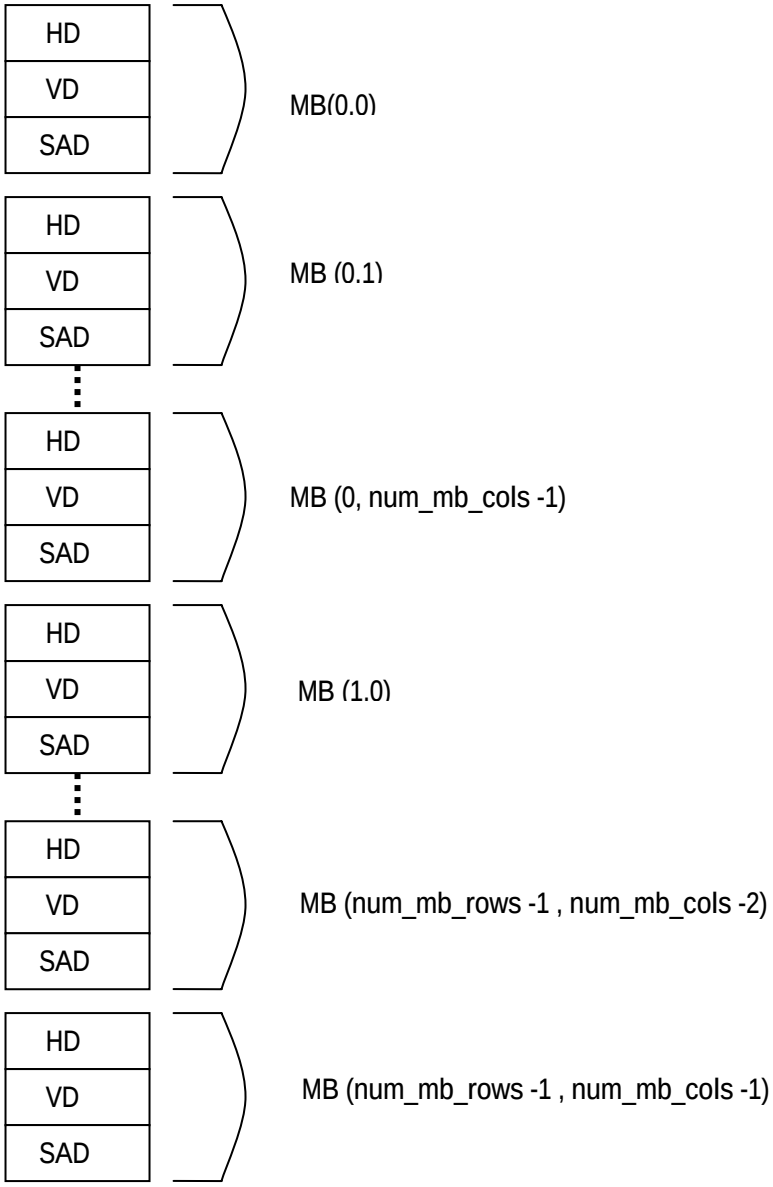


Figure 3-2. Motion Vector and SAD Buffer Organization.

3.4.2 Usage

The following sequence should be followed for motion vector access:

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

```
/* This structure defines the run-time parameters for
MP4VEnc object */

MP4VENC_DynamicParams   ext_dynamicParams;

/* Enable MV access */

ext_dynamicParams ->mvDataEnable = 1;

/* Control call to set the dynamic parameters */

control(.., XDM_SETPARAMS,..)
```

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

```
/* Output Buffer Descriptor variables */

XDM_BufDesc  outputBufDesc;

/* Get the input and output buffer requirements for the
codec */

control(.., XDM_GETBUFINFO, extn_dynamicParams, ..);
```

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

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

outputBufDesc.numBufs =
status.videncStatus.bufInfo.minNumOutBufs;

/* Stream Buffer */

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

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

/* MV Buffer */

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

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

3) Call the frame encode API:

```
/* Process call to encode 1 frame */

process(..  ,..  , outputBufDesc, .. );
```

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

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

Define a structure as shown.

```
struct motion mbdata
{
       short MVx;
       short MVy;
       unsigned int SAD;
} ;
motion mbdata  *mbMV data =  outputBufDesc.bufs[1];
num mb rows = frameRows / 16;
num mb cols  = frameCols / 16;
for (i = 0; i < num mb rows; i++)
{
      for (j = 0; j < num mb cols; j++)
     {
 HD for mb(i, j) = mbMV data ->MVx;
VD for mb(i, j) = mbMV data ->MVy;
SAD for mb(i,j) = mbMV data ->SSE;
mbMV data ++;
      }
}
```

Note:

- The motion vectors are with fullpel (integer pel) resolution.
- SSE = (Ref(i,j) – Src(i,j)) ^{2}, where Ref is the macro block of the reference region and Src is the macro block of the source image.
- Current version of the MPEG4 encoder stores the SAD (Sum of Absolute differences) in place of SSE.
- The motion vectors seen in the encoded stream are based on the best coding decision, which is a combination of the motion estimation and mode decisions. The MV buffer returns the results of the motion estimation in fullpel resolution (lowest SAD), which may be different from the motion vectors seen in the bit-stream:
  - Some macro blocks in a P-frame may be coded as Intra macro blocks based on the post motion estimation decisions. In this case, the motion vectors computed in the motion estimation stage (assuming that this macro block is

inter) will be returned.

- Some macro blocks in a P-frame may be 'Not Coded', that is, Skipped. In this case, motion vectors of (0,0) and SAD corresponding to (0,0) motion vector are returned.
- For I-frames, motion vectors are not returned and `outArgs.mvDataSize` = 0.

3.5 Accessing Reconstruction Buffer Data

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

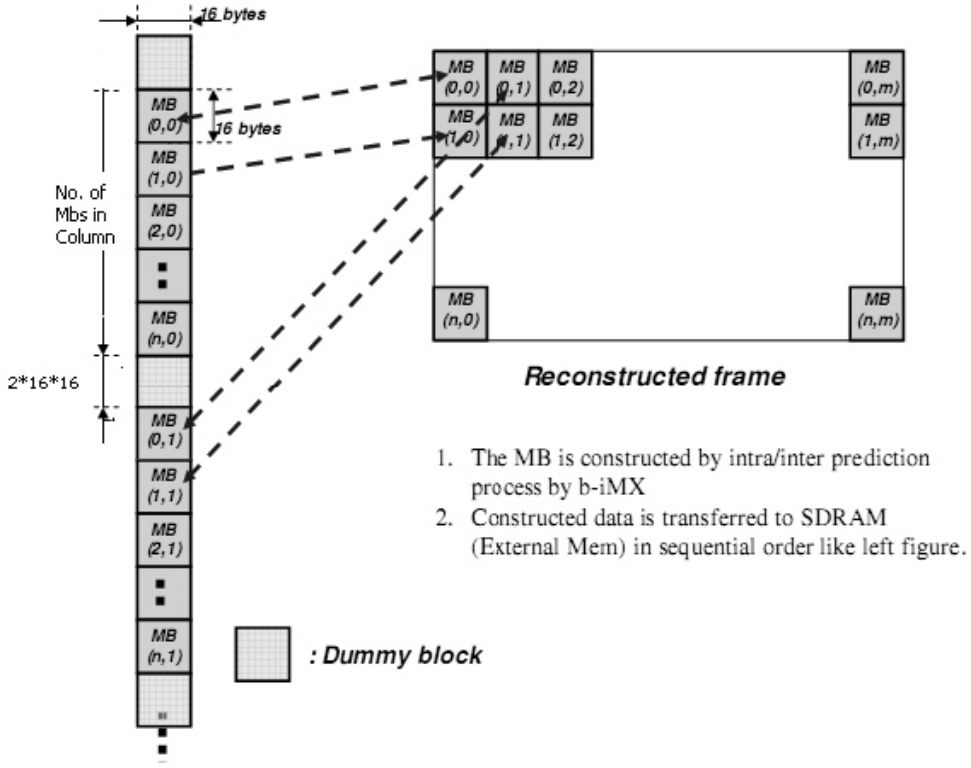


Figure 3-3. Reconstruction Buffer.

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

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

Following figures provides the format of the recon buffer taking SXVGA as an example.

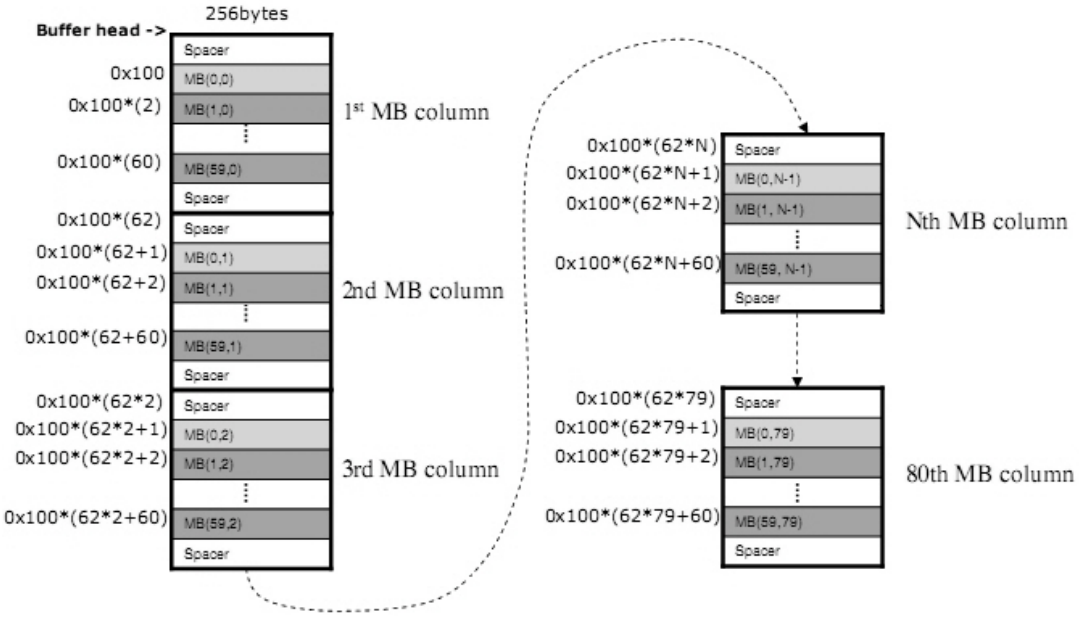


Figure 3-4. Reconstruction Buffer for Luma.

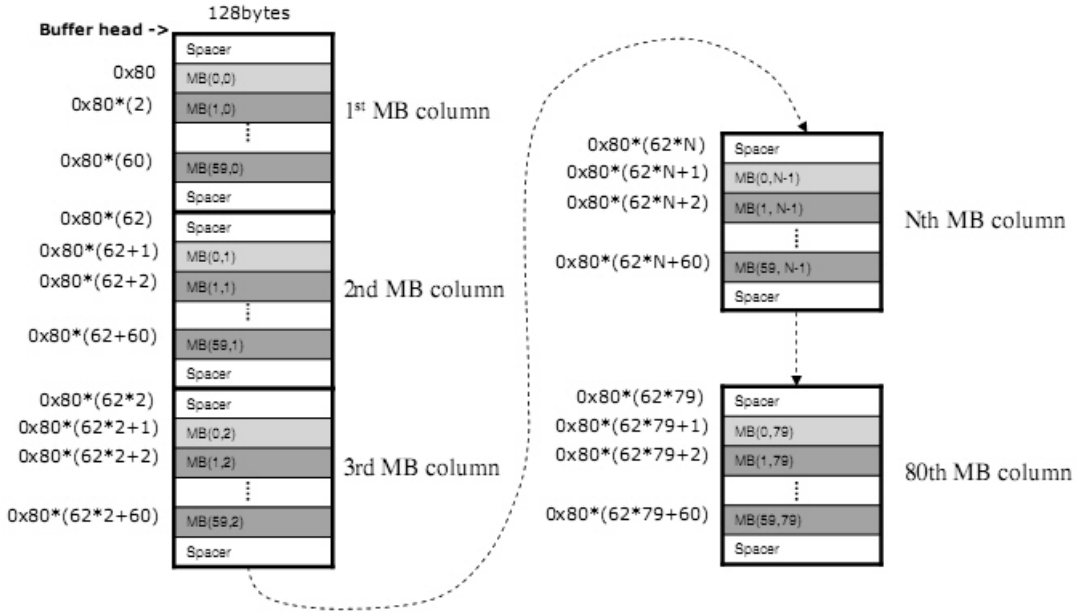


Figure 3-5. Reconstruction Buffer for Chroma.

A sample application code is provided as shown to extract luma and chroma data from the recon buffer and dump into output file in the planar YUV420 format.

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

/* Extract Luma*/
       for(k=0;k<16;k++)
       {
      for(l=0;l<16;l++)
      {
          temp_buffer[k*extn_params.subWindowWidth+l] = temp_lum[k*16+l];
      }
       }
        /* Extract Chroma*/
       for(k=0;k<8;k++)
       {
      for(l=0;l<8;l++)
      {
          temp_buffer_cb[k*(extn_params.subWindowWidth >> 1) + l] =
temp_chr[k*16+2*l];
          temp_buffer_cr[k*(extn_params.subWindowWidth >> 1) + l] =
temp_chr[k*16+(2*l)+1];
      }
        }
     }
}
fwrite(temp_buffer_bk, 1, ((extn_params.subWindowWidth *
extn_params.subWindowHeight *3)>>1), fReconBuffer);
..............
..............
```

3.6 User Data Insertion

MPEG4 Encoder provides the APIs to insert the User Data in encoded bit-stream. This can be used to incorporate useful information in the bit-stream. This User Data is inserted frame by frame.

The following figure shows how the bit-stream is built with the `UserData` field.

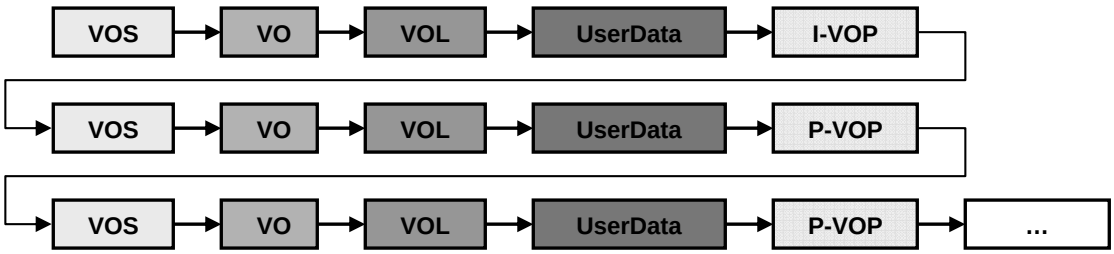


Figure 3-6. Bit-stream Built With UserData Field.

To support this feature, the following protocol is used between the codec and test application.

1) Codec will make space for user data insertion.
2) The start code insertion will be done by codec and specific user data will be inserted by the application. Codec will not insert the actual user data in the bit-stream.
3) Application will use `InArgs` to pass information about:
   - o User data is/is not
   - o Size of the data
4) Codec will use `OutArgs` to pass the offset in the output bit-stream where the user data should be inserted
5) The user data size sent to codec does not include the user data start code.

Application needs to set the following parameters of `IMP4VENC_InArgs` structure (in extended mode) before calling the `process` function.

Parameter Name	Type	Value (Range)
`insertUserData`	`Bool`	❑ 0 = Do not insert user data ❑ 1 = Insert user data
`lenghtUserData`	`UINT32`	❑ >0 (bytes): When `insertUserData` = 1 ❑ =0: When `insertUserData` = 0 **Error cases** ❑ >0 but `insertUserData` = 0, codec will assume that no user data

Parameter Name	Type	Value (Range)
		needs to be inserted ❑ =0 but `insertUserData` = 1, codec will assume that no user data needs to be inserted

After encoding process, codec will return the following parameters in `IMP4VENC_OutArgs` structure (in extended mode), which can be used by testApp to insert the user data.

Parameter Name	Type	Value (Range)
`offsetUserData`	`INT32`	❑ >=0 (bytes), Valid offset value when `insertUserData` = 1 ❑ =-1 , Value set by codec when `insertUserData` = 0, no space for user data insertion The offset (bytes) is with respect to the output buffer where the encoded frame is dumped after the `process()` call. Application should move to this offset and place the user data of `lengthUserData`.

This page is intentionally left blank

Chapter 4

API Reference

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

4.1 Symbolic Constants and Enumerated Data Types

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

Table 4-1. List of Enumerated Data Types.

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
IVIDEO_FrameType	IVIDEO_I_FRAME	0	Intra coded frame
	IVIDEO_P_FRAME	1	Forward inter coded frame
	IVIDEO_B_FRAME	2	Bi-directional inter coded frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_IDR_FRAME	3	Intra coded frame that can be used for refreshing video content. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_II_FRAME	4	Interlaced Frame, both fields are I frames. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_IP_FRAME	5	Interlaced Frame, first field is an I frame, second field is a P frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_IB_FRAME	6	Interlaced Frame, first field is an I frame, second field is a B frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_PI_FRAME	7	Interlaced Frame, first field is a P frame, second field is a I frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_PP_FRAME	8	Interlaced Frame, both fields are P frames. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_PB_FRAME	9	Interlaced Frame, first field is a P frame, second field is a B frame. Not supported in this version of the MPEG4 Encoder.

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
	IVIDEO_BI_FRAME	10	Interlaced Frame, first field is a B frame, second field is an I frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_BP_FRAME	11	Interlaced Frame, first field is a B frame, second field is a P frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_BB_FRAME	12	Interlaced Frame, both fields are B frames. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_MBAFF_I_FRAME	13	Intra coded MBAFF frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_MBAFF_P_FRAME	14	Forward inter coded MBAFF frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_MBAFF_B_FRAME	15	Bi-directional inter coded MBAFF frame. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_MBAFF_IDR_FRAME	16	Intra coded MBAFF frame that can be used for refreshing video content. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_FRAMETYPE_DEFAULT	0	Default set to IVIDEO_I_FRAME
IVIDEO_ContentType	IVIDEO_CONTENTTYPE_NA	-1	Content type is not applicable. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_PROGRESSIVE	0	Progressive video content. Default value.
	IVIDEO_INTERLACED	1	Interlaced video content. Not supported in this version of the MPEG4 Encoder.
IVIDEO_RateControlPreset	IVIDEO_LOW_DELAY	1	Constant Bit Rate (CBR) control for video conferencing. (that is, CBR with frames dropped based on VBV buffer occupancy)

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
	IVIDEO_STORAGE	2	Variable Bit Rate control for local storage (DVD) recording. Default value (similar to IVIDEO_LOW_DELAY but no frame skips) (See section 4.2.2)
	IVIDEO_TWOPASS	3	Two pass rate control for non real time applications. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_NONE	4	No rate control algorithm (Fixed Qp values)
	IVIDEO_USER_DEFINED	5	User defined through extended parameters. Not supported in this version of the MPEG4 Encoder.
	IVIDEO_RATECONTROLPRESET_DEFAULT	1	Set to IVIDEO_LOW_DELAY
IVIDEO_SkipMode	IVIDEO_FRAME_ENCODED	0	Input content encoded
	IVIDEO_FRAME_SKIPPED	1	Input content skipped, that is, not encoded
	IVIDEO_SKIPMODE_DEFAULT	0	Default value set to IVIDEO_FRAME_ENCODE
XDM_DataFormat	XDM_BYTE	1	Big endian stream
	XDM_LE_16	2	16-bit little endian stream. Not supported in this version of the MPEG4 Encoder.
	XDM_LE_32	3	32-bit little endian stream. Not supported in this version of the MPEG4 Encoder.
XDM_ChromaFormat	XDM_CHROMA_NA	-1	Chroma format not applicable. Not supported in this version of the MPEG4 Encoder.
	XDM_YUV_420P	1	YUV 4:2:0 planar
	XDM_YUV_422P	2	YUV 4:2:2 planar. Not supported in this version of the MPEG4 Encoder.

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
	XDM_YUV_422IBE	3	YUV 4:2:2 interleaved (big endian). Not supported in this version of the MPEG4 Encoder.
	XDM_YUV_422ILE	4	YUV 4:2:2 interleaved (little endian)
	XDM_YUV_444P	5	YUV 4:4:4 planar. Not supported in this version of the MPEG4 Encoder.
	XDM_YUV_411P	6	YUV 4:1:1 planar. Not supported in this version of the MPEG4 Encoder.
	XDM_GRAY	7	Gray format. Not supported in this version of the MPEG4 Encoder.
	XDM_RGB	8	RGB color format. Not supported in this version of the MPEG4 Encoder.
	XDM_YUV_420SP	9	YUV 4:2:0 semi planar (Yplane, Cb Cr plane)
XDM_CmdId	XDM_GETSTATUS	0	Query algorithm instance to fill `Status` structure
	XDM_SETPARAMS	1	Set run-time dynamic parameters through the `DynamicParams` structure
	XDM_RESET	2	Reset the algorithm.
	XDM_SETDEFAULT	3	Initialize all fields in `Params` structure to default values specified in the library
	XDM_FLUSH	4	Handle end of stream conditions. This command forces algorithm instance to output data without additional input.
	XDM_GETBUFINFO	5	Query algorithm instance regarding the properties of input and output buffers
	XDM_GETVERSION	6	Query the algorithm's version. Not supported in this version of the MPEG4 Encoder
XDM_EncodingPreset	XDM_DEFAULT	0	Default setting of the algorithm specific creation time parameters. This uses `XDM_HIGH_QUALITY` settings

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

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:

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

4.2.1.1 XDM1_BufDesc

‖ Description

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

‖ Fields

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

4.2.1.2 XDM1_SingleBufDesc

Description

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

Fields

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

4.2.1.3 XDM1_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

4.2.1.4 IVIDEO1_BufDesc

Description

This structure defines the buffer descriptor for input video buffers.

Fields

Field	Data Type	Input/ Output	Description
numBufs	XDAS_Int32	Input	Number of buffers in bufDesc[]
frameWidth	XDAS_Int32	Input	Width of the video frame
frameHeight	XDAS_Int32	Input	Height of the video frame
framePitch	XDAS_Int32	Input	Frame pitch used to store the frame. Not Supported in this version of MPEG4 encoder
bufDesc[XDM_MAX_IO_BUFFERS]	XDM1_SinglBufDesc	Input	Picture buffers

4.2.1.5 IVIDENC1_Fxns

Description

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

Fields

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

4.2.1.6 IVIDENC1_Params

‖ Description

This structure defines the creation parameters 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.
encodingPreset	XDAS_Int32	Input	Encoding preset. See XDM_EncodingPreset enumeration for details. Only the following values are supported: ❑ XDM_HIGH_QUALITY (1): For Very High Quality and Low Performance. This is same as XDM_DEFAULT (0). ❑ XDM_HIGH_SPEED(2): For Moderate Quality and Moderate Performance. ❑ XDM_USER_DEFINED(3): For using the user-defined extended parameters. The default is XDM_HIGH_QUALITY. In addition to this, the encoder supports the following enumerations: ❑ IMP4VENC_HIGH_QUALITY_MODERATE_PERFORMANCE (256): For High Quality and Moderate Performance. ❑ IMP4VENC_HIGHEST_QUALITY_LOWEST_PERFORMANCE (257): For Highest Quality and Lowest Performance. ❑ IMP4VENC_LOW_QUALITY_HIGHEST_PERFORMANCE (258): For Normal Quality and Highest Performance. **Note**: encodingPreset must be set to XDM_USER_DEFINED, if you need to set quality/performance tools through extended parameters.

Field	Data Type	Input/ Output	Description
rateControlPreset	XDAS_Int32	Input	Rate control preset: See IVIDEO_RateControlPreset enumeration for details. Only IVIDEO_LOW_DELAY, IVIDEO_STORAGE and IVIDEO_NONE are supported. The default is IVIDEO_LOW_DELAY. The relevant extended parameters should be set to use any of these rate controls. (See section 4.2.2) Apart from this, following additional rate control algorithm is supported ❑ IMP4VENC_CVBR_LBR1: Advanced CVBR algorithm suitable for low bit-rate. Does not allow frame skips. ❑ IMP4VENC_CVBR_LBR2: Advanced CVBR algorithm suitable for low bit-rate. Allows frame skips. ❑ IMP4VENC_CVBR: Advanced CVBR algorithm
maxHeight	XDAS_Int32	Input	Height of the input stream in pixels. Default: 960 Supported Value: 64 to 1920
maxWidth	XDAS_Int32	Input	Width of the input stream in pixels. Default: 1280 Supported Value: 160 to 1920.
maxFrameRate	XDAS_Int32	Input	Frame rate in fps * 1000. Default: 30000 Supported Value: 1000 to 30000000
maxBitRate	XDAS_Int32	Input	Maximum Bit-rate to be used for encoding in bits per second. Maximum supported value is 51000000.
dataEndianness	XDAS_Int32	Input	Endianness of input data. See XDM_DataFormat enumeration for details. Only XDM_BYTE is supported in this version.
maxInterFrameInterval	XDAS_Int32	Input	Distance from I-frame to P-frame. Default : 1 Supported Value: 0, 1
inputChromaFormat	XDAS_Int32	Input	Input chroma format. See XDM_ChromaFormat enumeration for details. Only XDM_YUV_422ILE (4) and XDM_YUV_420SP (9) are supported in this version. The default is XDM_YUV_422ILE.
inputContentType	XDAS_Int32	Input	Input content type. See IVIDEO_ContentType enumeration for details. Only IVIDEO_PROGRESSIVE is supported in this version. The default is IVIDEO_PROGRESSIVE.

Field	Data Type	Input/ Output	Description
reconChromaFormat	XDAS_Int32	Input	Chroma formats for the reconstruction buffers. Reconstruction buffer chroma format is neither exactly YUV 4:2:0 P nor 4:2:0 SP. See section 3.5, for more details.

4.2.1.7 IVIDENC1_DynamicParams

Description

This structure defines the run-time parameters 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.
inputHeight	XDAS_Int32	Input	Height of input frame in pixels. Must be multiple of 16. Default: 480 Supported Value: 64 to 1920 Note: Though supported value is upto 1920, codec does not give error until 2048. This is to support encoding of (up to) 1080p even if video is captured at higher resolution (height upto 2048)
inputWidth	XDAS_Int32	Input	Width of input frame in pixels. Must be multiple of 16. Default: 720 Supported Value: 160 to 1920 (with UMV OFF) and 192 to 1920 (with UMV ON) Note: Though supported value is upto 1920, codec does not give error until 2048. This is to support encoding of (up to) 1080p even if video is captured at higher resolution (height upto 2048) .
refFrameRate	XDAS_Int32	Input	Reference or input frame rate in fps. Not supported in this version of the encoder.
targetFrameRate	XDAS_Int32	Input	Target frame rate in fps * 1000. Default: 30000 Supported Value : 1000 to 30000000 (See section 4.2.3)
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. (see section 4.2.3). Maximum supported value is 51000000.

Field	Data Type	Input/ Output	Description
intraFrameInterval	XDAS_Int32	Input	The number of frames between two I frames. Default: 30 Supported Value : 1 to 100 ❑ 1 : No inter frames (all intra frames) ❑ n : n-1 frames coded as p-frames between every two I-frames (see section 4.2.3)
generateHeader	XDAS_Int32	Input	Encode entire access unit or only header. The following values are supported: ❑ XDM_ENCODE_AU (0): Encode entire access unit, including the headers. (Default) ❑ XDM_GENERATE_HEADER(1): Encode only header.
captureWidth	XDAS_Int32	Input	If the field is set to: ❑ 0 : Encoded image width is used as pitch. ❑ Any non-zero value, capture width is used as pitch (if capture width is greater than image width). Not supported in this version of the encoder.
forceFrame	XDAS_Int32	Input	Force the current (immediate) frame to be encoded as a specific frame type. Only IVIDEO_I_FRAME and IVIDEO_NA_FRAME is supported. (see section 4.2.3)
interFrameInterval	XDAS_Int32	Input	Number of B frames between two reference frames; that is, the number of B frames between two P frames or I/P frames. Since B frame is not supported, only value of 0 (default) and 1 is supported for this parameter.
mbDataFlag	XDAS_Int32	Input	Flag to indicate that the algorithm should use MB data supplied in additional buffer within inBufs. Not supported in this version of the encoder.

4.2.1.8 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.

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

4.2.1.9 IVIDENC1_Status

Description

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

Fields

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

4.2.1.10 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 after each process/encode call
encodedFrameType	XDAS_Int32	Output	Frame types for video. See IVIDEO_FrameType enumeration for details. Only IVIDEO_I_FRAME, IVIDEO_P_FRAME are supported in this version.
inputFrameSkip	XDAS_Int32	Output	Frame skipping modes for video. See IVIDEO_SkipMode enumeration for details.
outputID	XDAS_Int32	Output	Identifier to attach with the corresponding encoded bit-stream frames.
encodedBuf	XDM1_SingleBufDesc	Output	Not supported in this version.
reconBufs	IVIDEO1_BufDesc	Output	Pointer to reconstruction buffer descriptor. See section 3.5, for more details.

4.2.2 IVIDEO_RateControlPreset

MPEG4 encoder supports the following types of rate control algorithms:

- IVIDEO_STORAGE (VBR)
- IVIDEO_LOW_DELAY(CBR)
- IVIDEO_NONE (fixed Qp)
- IMP4VENC_CVBR_LBR1
- IMP4VENC_CVBR_LBR2
- IMP4VENC_CVBR

The behavior of rate control algorithm is decided by the following parameters (see IMP4VENC_Params and IMP4VENC_DynamicParams structure for details):

- initQ
- rcQMax
- rcQMin
- intraFrameQp
- interFrameQp
- vbvSize
- IFrameBitRateBiasFactor
- PFrameBitRateBiasFactor
- peakBufWindow
- minBitRate
- maxBitRate

  targetBitRate

4.2.2.1 IVIDEO_STORAGE (VBR)

- If IVIDEO_RateControlPreset is set to IVIDEO_STORAGE, it selects the Variable Bit rate.
- The parameter initQ decides the starting quantization parameter for I frames. Qp will change at MB level.
- The achieved bit-rate may not necessarily be close to the specified bit-rate, as frame skip is not allowed.
- The vbvSize parameter is ignored for VBR.

4.2.2.2 IVIDEO_LOW_DELAY(CBR)

- If `IVIDEO_RateControlPreset` is set to `IVIDEO_LOW_DELAY`, it selects the Constrained Bit rate (CBR).
- The parameters `initQ` decides the starting quantization parameter for I frame. `Qp` will change at MB level.
- The `vbvSize` parameter is considered for CBR.

  A typical value for `vbvSize` can be obtained from the following equation:

  `vbvSize=` BitRate / ((FrameRate/ 1000) * 16000).

  CBR models the encoder decoder coupled bit buffer memory modeling. The encoder is assumed to fill the bits in the buffer and decoder is assumed to drain out the bits from the same buffer. The size of this buffer is the size of `vbvSize` parameter. The encoder maps the buffer into which it is writing bits, by mapping it through VBV position variable. Once the buffer gets filled, the encoder is forced to skip the frame, thereby preventing the possible corruption of encoded bit buffer.

  The initial VBV position is set as 50% and this corresponds to the initial buffer fullness assumed at the decoder side. The VBV position is updated based on the logic mentioned below. If the VBV position reaches VBV size, it implies that encoded bits buffer is full and frame is skipped. This decreases the VBV position as decoder drains out the bits continuously.

  ```
  VBV_pos = (0.5 x vbvSizex 16000)
  DrainRate  = (BitRate/FrameRate)
  ```

  After every frame encoding, VBV position is up dated, and skip decision is made as follows.

  ```
  SkipFrame  = 0;
  VBV_pos += (No of Bits encoded - DrainRate)
  If (VBV_pos > (VBV_size * 16000))
  {
     SkipFrame = 1;
  }
  ```

  The dynamically varied `bitRate` and `frameRate` is considered in VBV modeling, without reinitializing the VBV buffer.

  The amount of variation in the achieved bit-rate is proportional to the VBV buffer size. Very less VBV size will yield less variation in the achieved bit-rate (compared to target bit-rate) but will have more number of frames skipped.

4.2.2.3 Fixed Qp (IVIDEO_NONE)

- If `IVIDEO_RateControlPreset` is set to `IVIDEO_NONE`, it selects the no rate control mode.
- The parameters `intraFrameQp` and `interFrameQp` decides the quantization parameter for I and P frames, respectively. `Qp` will not change at MB level.

- All the Macroblocks of I frame are coded with `intraFrameQp` and all Macroblocks of P frame are coded with `interFrameQp`.
- The `bitRate` parameter is ignored, so achieved bit-rate may not be same as the specified bit-rate.
- If you set `intraFrameQp` and `interFrameQp` parameters as `XDM_DEFAULT` (that is, zero), these parameters will be calculated by codec.
- The vbvSize is ignored.

4.2.2.4 IMP4VENC_CVBR_LBR1

- If `IVIDEO_RateControlPreset` is set to `IMP4VENC_CVBR_LBR1`, it selects the Constrained Low Bit rate (LBR1).
- The behavior of this algorithm is similar to that of Variable Bit Rate (VBR). In addition it provides control (to user) to bias the bits allocation to I & P frame using following parameters.

  `IFrameBitRateBiasFactor:` Bits allocated to I frame will be biased by a factor of (`IFrameBitRateBiasFactor/256)`

  `PFrameBitRateBiasFactor:` Bits allocated to P frame will be biased by a factor of (`PFrameBitRateBiasFactor/256)`

  1. Default values to these parameters are 256 and 256 for “No Bias” case.
  2. Changing these values for Positive bias(>256) or negative bias (<256) will have impact on achieved bitrate.
  3. Recommended settings for these parameters are 512 and 256 respectively for very low bitrate.

- This algorithm does not allow the frame skips. Hence there might be deviation between achieved bitrate and target bitrate incase targetbitrate is not achieved even with lowest quality (Qp =31).

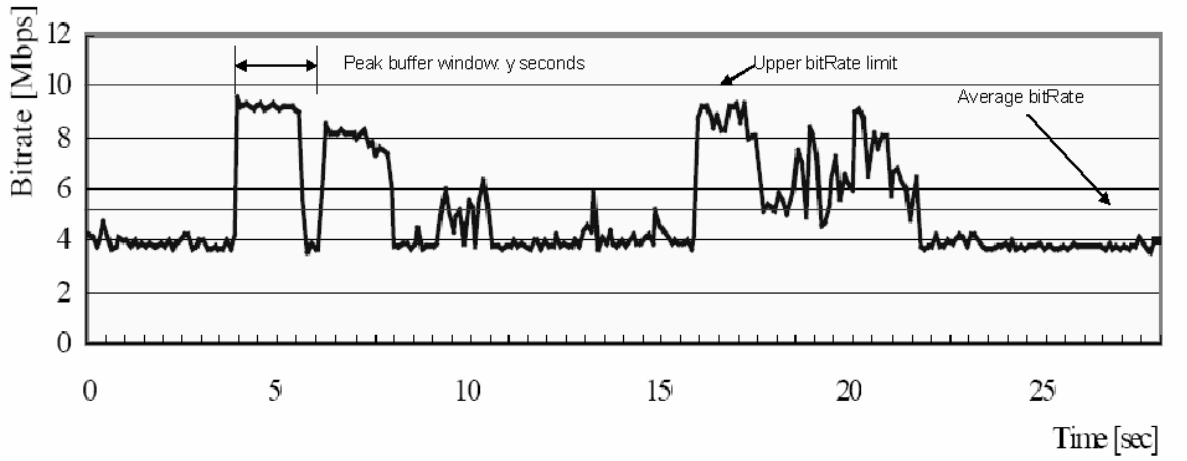
4.2.2.5 IMP4VENC_CVBR_LBR2

- If `IVIDEO_RateControlPreset` is set to `IMP4VENC_CVBR_LBR2`, it selects the Constrained Low Bit rate (LBR2).
- The behavior of this algorithm is similar to that of `IMP4VENC_CVBR_LBR1` except that it allows the “frame skipping” to meet the target bitrate.

4.2.2.6 IMP4VENC_CVBR

- If `IVIDEO_RateControlPreset` is set to `IMP4VENC_CVBR`, it selects the Constrained Variable Bit rate (CVBR).
- Keeps reasonable video quality when video scene is complicated by setting up appropriate upper bitrate limit (maxBitRate) within the peak buffer window (peakBufWindow).

- Reduces actual bitrate consumption down to optional minimum bitrate (minBitRate) to achieve specified average bitrate when video scene is easy.
- Avoids frame skips.



4.2.3 Usage of Dynamic Parameters

4.2.3.1 IntraFrameInterval

When `intraFrameInterval` is changed at run-time by `SETPARAMS`, the immediate next frame will be forced to I frame and a new GOP will start. The new `intraFrameInterval` will be effective from the immediate next frame (which is forced to be an I frame).

Following diagram explains the usage, when `intraFrameInterval` = 30 at create time and has been changed to 10 (through `SETPARAMS` API) after encoding 48 frames.

I	I frame
-	P frame
|<--------->|	one second duration
|----------|	GOP

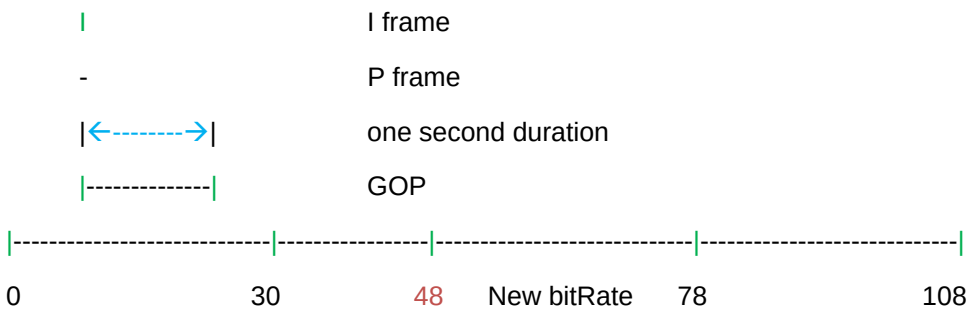
```
|-----------------------------|-----------------|---------|---------|---------|---------|---------|---------|
0                             30                48        58        68        78        88        98        108
```

4.2.3.2 TargetBitRate

When `targetBitRate` is changed at run-time by `SETPARAMS`, the immediate next frame will be forced to I frame and a new GOP will start.

The new bit-rate will be effective from the immediate next frame (which is forced to be an I frame).

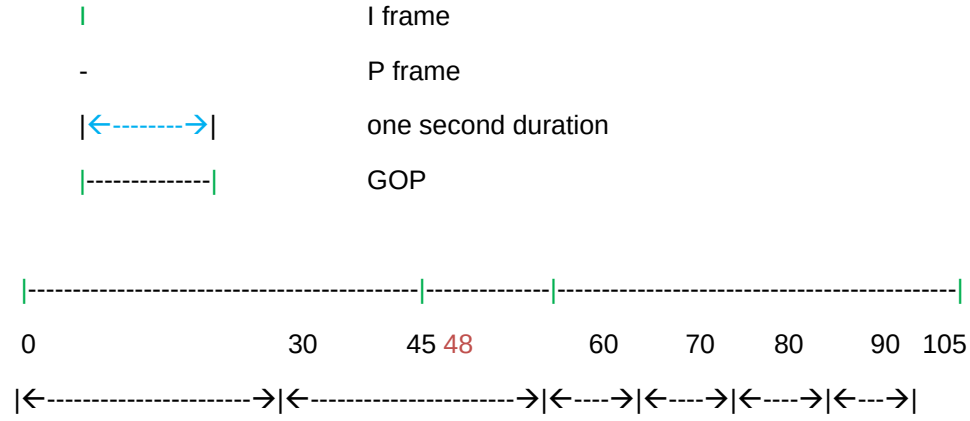
Following diagram explains the usage, `targetBitRate` has been changed (through `SETPARAMS` API) after encoding 48 frames.



> **Note:**
>
> `vbv_buffer_size` is not a dynamic parameter. Even if the `bitRate` changes, this will remain same as set during algorithm creation.

4.2.3.3 TargetFrameRate

When `targetFrameRate` is changed at run-time by `SETPARAMS`, the immediate next second starting frame will be force to I frame and a new GOP will start. The new frame rate will be effective from next second.



4.2.3.4 forceFrame

Force the current (immediate) frame to be encoded as a specific frame type. Only `IVIDEO_I_FRAME` is supported. Force the immediate frame to be an I frame. Next I frame will be after the `intraFrameInterval` from current frame.

4.2.4 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
- IMP4VENC_ERROR

4.2.4.1 IMP4VENC_Params

Description

This structure defines the creation parameters and any other implementation specific parameters for the MPEG4 Encoder instance object. The base 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.
subWindowHeight	XDAS_Int32	Input	Height of the sub window must be multiple of 8 (codec does not have validity check and application needs to ensure that this validation is performed). It should be less than or equal to maxHeight specified in IVIDENC1_params. Supported Value: 64 to 1920 Note: User should set this parameter same as actual inputHeight, if subWindow feature is not used.
subWindowWidth	XDAS_Int32	Input	Width of the Subwindow, must be multiple of 16. Must be less than or equal to maxWidth specified in IVIDENC1_params. Supported Value: 160 to 1920 (with UMV OFF) and 192 to 1920 (with UMV ON) Note: User should set this parameter same as actual inputWidth, if subWindow feature is not used.

Field	Data Type	Input/ Output	Description
rotation	XDAS_Int32	Input	Rotation (anti-clockwise): ❑ 0: No Rotation (Default) ❑ 90: 90 degree ❑ 180: 180 degree ❑ 270: 270 degree Other values are not supported.
vbvSize	XDAS_Int32	Input	Video buffer verifier size in 16 kb. Default: 0 (codec will internally calculate the value) **Note**: This parameter's value is based on the resolution of the video.
svhMode	XDAS_Int32	Input	❑ 0: encode in mpeg4 mode ❑ 1: encode in mpeg4 with short video header mode
IFrameBitRateBiasFactor	XDAS_Int32	Input	This parameter controls the biasing of bits allocated to I frame. Range : 1 to 1024 Default: 256, i.e. No bias Note: This parameter must be set to a valid value if rateControlPreset is set to IMP4VENC_CVBR_LBR1 or IMP4VENC_CVBR_LBR2
PFrameBitRateBiasFactor	XDAS_Int32	Input	This parameter controls the biasing of bits allocated to P frame. Range : 1 to 1024 Default: 256, i.e. No bias Note: This parameter must be set to a valid value if rateControlPreset is set to IMP4VENC_CVBR_LBR1 or IMP4VENC_CVBR_LBR2
peakBufWindow	XDAS_Int32	Input	Time duration (in sec) during which actual bitrate of encoding stream can reach the max bitrate limit Default: 2 Note: This parameter must be set to a valid value if rateControlPreset is set to IMP4VENC_CVBR
minBitRate	XDAS_Int32	Input	Minimum bit rate, bits per second. Default: to set this to same as targetBitRate. Note: This parameter must be set to a valid value if rateControlPreset is set to IMP4VENC_CVBR

4.2.4.2 IMP4VENC_DynamicParams

‖ Description

This structure defines the run-time parameters and any other implementation specific parameters for an MPEG4 Encoder instance

object. The base dynamic parameters are defined in the XDM data structure, `IVIDENC1_DynamicParams`..

‖ Fields

Field	Data Type	Input/ Output	Description
`videncDynamicparams`	`IVIDENC1_DynamicParams`	Input	See `IVIDENC1_DynamicParams` data structure for details
`intraAlgo`	`XDAS_Int32`	Input	INTRA/INTER Decision Algorithm. ❑ `IMP4VENC_INTRA_INTER_DECISION_LQ_HP (0)`: for low quality and high performance (Default) ❑ `IMP4VENC_INTRA_INTER_DECISION_HQ_LP (1)`: for high quality and low performance
`numMBRows`	`XDAS_Int32`	Input	Number of MB rows in a Packet. ❑ Maximum value = subWindowHeight/16. This indicates the packet size (Default). ❑ Minimum value = 1.
`initQ`	`XDAS_Int32`	Input	Initial Q (at picture head). ❑ 0: automatically determined (Default) ❑ 1-31: Force initial Q. This is for I frame quantization. This should have a value between `rcQMax` and `rcQmin`, if not set to 0.
`rcQMax`	`XDAS_Int32`	Input	Q MAX value ❑ Maximum value = 31 (Default) ❑ Minimum value = 1
`rcQMin`	`XDAS_Int32`	Input	Q MIN value ❑ Maximum value = 31 ❑ Minimum value = 1 (Default)
`intraFrameQp`	`XDAS_Int32`	Input	I Frame Qp ❑ 0: automatically determined (Default) ❑ 1-31: Force I Frame Qp. This should have a value between `rcQMax` and `rcQmin`, if not set to 0.

Field	Data Type	Input/ Output	Description
interFrameQp	XDAS_Int32	Input	P Frame Qp ❑ 0: automatically determined (Default) ❑ 1-31: Force P Frame Qp. This should have a value between rcQMax and rcQmin, if not set to 0.
rateFix	XDAS_Int32	Input	Reserved. Codec ignores the value of this parameter.
rateFixRange	XDAS_Int32	Input	Reserved. Codec ignores the value of this parameter.
meAlgo	XDAS_Int32	Input	Motion estimation algorithm ❑ IMP4VENC_ME_MQ_MP (0): For moderate quality and moderate performance. ❑ IMP4VENC_ME_HQ_MP (1): For high quality and moderate performance. ❑ IMP4VENC_ME_HQ_LP (2): For highest quality and lowest performance. (Default) ❑ IMP4VENC_ME_LQ_HP (3): For normal quality and highest performance.
skipMBAlgo	XDAS_Int32	Input	P Skip MB algorithm ❑ IMP4VENC_SKIP_MB_LQ_HP (0): Non-Bonus Skip MB,for low quality and high performance (Default) ❑ IMP4VENC_SKIP_MB_HQ_LP (1): Bonus SKIP MB , for high quality and low performance.
unrestrictedMV	XDAS_Int32	Input	Unrestricted motion vector ❑ IMP4VENC_UMV_LQ_HP (0): disable (Default) ❑ IMP4VENC_UMV_HQ_LP(1): enable **Note**: If UMV is enabled, the minimum input stream width supported is 192.
mvDataEnable	XDAS_Int32	Input	❑ 0: Disable motion vector access (Default) ❑ 1: Enable motion vector access

4.2.4.3 IMP4VENC_InArgs

‖ Description

This structure defines the input argument parameters and any other implementation specific parameters for the MPEG4 Encoder instance object. The base input parameters are defined in XDM data structure, `IVIDENC1_InArgs`.

‖ Fields

Field	Data Type	Input/ Output	Description
`videnc_InArgs`	`IVIDENC1_InArgs`	Input	See `IVIDENC1_InArgs` data structure for details.
`subWindowHozOfst`	`XDAS_Int32`	Input	Horizontal Offset of the `Subwindow` from the input image. Not supported in this version
`subWindowVerOfst`	`XDAS_Int32`	Input	Vertical Offset of the Subwindow from the input image. Not supported in this version
`insertUserData`	`Bool`	Input	❑ 0 = Do not insert user data (Default) ❑ 1 = Insert user data
`lenghtUserData`	`UINT32`	Input	❑ >0 (bytes). when `insertUserData` = 1 ❑ =0, when `insertUserData` = 0 **Error cases** ❑ >0 but `insertUserData` = 0, codec will assume that no user data needs to be inserted ❑ =0 but `insertUserData` = 1, codec will assume that no user data needs to be inserted

4.2.4.4 IMP4VENC_Status

‖ Description

This structure defines parameters that describe the status of the MPEG4 Encoder and any other implementation specific parameters. The base 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.

4.2.4.5 IMP4VENC_OutArgs

‖ Description

This structure defines the output arguments for the MPEG4 Encoder instance object. The base output parameters are defined in XDM data structure, `IVIDENC1_OutArgs`.

‖ Fields

Field	Data Type	Input/ Output	Description
`videnc_OutArgs`	`IVIDENC1_OutArgs`	Output	See `IVIDENC1_OutArgs` data structure for details.
`mvDataSize`	`XDAS_Int32`	Output	Size of the motion vector array.
`offsetUserData`	`XDAS_Int32`	Output	❑ >=0 (bytes), Valid offset value when `insertUserData` = 1 ❑ =-1 , Value set by codec when `insertUserData` = 0, no space for user data insertion The offset (bytes) is with respect to the output buffer where the encoded frame is dumped after the `process()` call. Application should move to this offset and put the user data of `lengthUserData`.

4.2.4.6 IMP4VENC_ERROR

‖ Description

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

‖ Fields

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
IMP4VENC_ERROR	IMP4VENC_INVALID_IMAGEWIDTH	1	Invalid image width
	IMP4VENC_INVALID_IMAGEHEIGHT	2	Invalid image height Ignore
	IMP4VENC_INVALID_ENCODINGPRESET	3	Invalid encoding preset
	IMP4VENC_INVALID_RATECONTROLPRESET	4	Invalid Rate control preset
	IMP4VENC_INVALID_MAXINTERFRMINTERVAL	5	Invalid maximum inter frame interval (if value is other than 1)
	IMP4VENC_INVALID_INPUTCONTENTTYPE	6	Invalid input content type
	IMP4VENC_INVALID_RECONCHROMAFORMAT	7	Invalid Recon chroma format
	IMP4VENC_INVALID_ROTATION	8	Invalid value of rotation
	IMP4VENC_INVALID_FRAMERATE	9	Invalid value of frame rate
	IMP4VENC_INVALID_BITRATE	10	Invalid value for bit-rate
	IMP4VENC_INVALID_INTRAFRAMEINTERVAL	11	Invalid value of intra frame interval
	IMP4VENC_INVALID_INTERFRAMEINTERVAL	12	Invalid value of inter frame interval
	IMP4VENC_INVALID_INTRAALGO	13	Invalid value of intra algo
	IMP4VENC_INVALID_NUMMBROWS	14	Invalid value of number of MB rows
	IMP4VENC_INVALID_INITQ	15	Invalid initial quantization value for I frame

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
	IMP4VENC_INVALID_RCQMAX	16	Invalid value of rcMAX parameter
	IMP4VENC_INVALID_RCQMIN	17	Invalid value of rcMIN parameter
	IMP4VENC_INVALID_RATEFIX	18	Invalid value of rate fix parameter
	IMP4VENC_INVALID_RATEFIXRANGE	19	Invalid value of rate fix range parameter
	IMP4VENC_INVALID_VBVSIZE	20	Invalid value of virtual buffer verifier parameter
	IMP4VENC_INVALID_MEALGO	21	Invalid value of ME algo parameter
	IMP4VENC_INVALID_SKIPMBALGO	22	Invalid value of skip MB algo parameter
	IMP4VENC_INVALID_UMV	23	Invalid value of UMV parameter
	IMP4VENC_INVALID_SVH	24	Invalid value of SVH parameter
	IMP4VENC_INVALID_FORCEFRAME	25	Invalid value for `forceFrame` parameter
	IMP4VENC_INVALID_GENERATEHEADER	26	Invalid value of generate header parameter
	IMP4VENC_INVALID_MVDATAENABLE	27	Invalid value of motion vector access parameter
	IMP4VENC_INVALID_INTRAFRAMEQP	28	Invalid Intra Frame Qp
	IMP4VENC_INVALID_INTERFRAMEQP	29	Invalid Inter Frame Qp

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()`, `dmaGetChannelCnt()`, `dmaGetChannels()`
- **Initialization** – `algInit()`, `dmaInit()`
- **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) `control()`
5) `algActivate()`
6) `process()`
7) `algDeactivate()`
8) `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.

> **Note**:
>
> See *MPEG4 Encoder Data Sheet* for more details on External Data Memory requirement.

‖ 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.

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

‖ See Also

`algAlloc(), algMoved()`

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

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

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

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

```
{
    ……….
    ……….

    VIDENC1_Params              params;
    IMP4VENC_Params             extParams;

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

    // Set the create time extended parameters

    extParams.videncParams = params;

    extParams.subWindowHeight = 480;
    extParams.subWindowWidth = 720;
            extParams.rotation = XDM_DEFAULT;

    extParams.vbvSize = 10000;
    extParams.svhMode = 0
```

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

4.3.3 Control API

Control API is used before a call to process() to enquire about the number and size of I/O buffers, or to set the dynamic params, or get status of encoding.

‖ 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.

‖ Post conditions

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

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

‖ Example

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

‖ See Also

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

The following code provides an example for initializing the base dynamic parameters for a 720x480 stream.

```
                    {
……….
……….
    VIDENC1_DynamicParams       dynParams;
    IVIDENC1_Status             status;
……….
……….

    // Set the dynamic base parameters
    dynParams.size = sizeof(VIDENC1_DynamicParams);
    dynParams.inputHeight = 480;
    dynParams.inputWidth = 720;
    dynParams.refFrameRate = FRAMERATE;
    dynParams.targetFrameRate = FRAMERATE;
    dynParams.targetBitRate = BITRATE;
    dynParams.intraFrameInterval = 30;
    dynParams.generateHeader = XDM_DEFAULT;
    dynParams.captureWidth = XDM_DEFAULT;
    dynParams.forceFrame = -1;
    dynParams.interFrameInterval = XDM_DEFAULT;
    dynParams.mbDataFlag = XDM_DEFAULT;

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

The following code gives an example for initializing the extended dynamic parameters for a 720x480 stream with out motion vector access.

```
                    {
……….
……….
    VIDENC1_DynamicParams dynParams;
    IVIDENC1_Status             status;
    IMP4VENC_DynamicParams      extDynParams;
……….
……….
    // Set the dynamic base parameters
    dynParams.size = sizeof(IMP4VENC_Params);
    dynParams.inputHeight = 480;
    dynParams.inputWidth = 720;
    dynParams.refFrameRate = FRAMERATE;
    dynParams.targetFrameRate = FRAMERATE;
    dynParams.targetBitRate = BITRATE;
    dynParams.intraFrameInterval = 30;
    dynParams.generateHeader = XDM_DEFAULT;
    dynParams.captureWidth = XDM_DEFAULT;
    dynParams.forceFrame = -1;
    dynParams.interFrameInterval = XDM_DEFAULT;
    dynParams.mbDataFlag = XDM_DEFAULT;
    // Set the extended dynamic parameters
    extDynParams.videncDynamicParams = dynParams;


    extDynParams.intraAlgo = 2;
    extDynParams.numMBRows = 5;
    extDynParams.initQ = 3;
    extDynParams.rcQMax = 31;
    extDynParams.rcQMin = 1;
    extDynParams.intraFrameQP = 0;
    extDynParams.interFrameQP = 0;
    extDynParams.rateFix = 0;
    extDynParams.rateFixRange = 0;

    extDynParams.meAlgo = 1;

    extDynParams.skipMBAlgo = XDM_DEFAULT;
    extDynParams.unrestrictedMV = XDM_DEFAULT;


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

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()`

The following sample code gives an example of process call.

```
{
……
……
retVal = ividEncfxns->process((IVIDENC1 Handle) handle,

                     (IVIDEO1 BufDescIn *)
                     &inputBufDesc,(XDM BufDesc *)
                    &outputBufDesc,
                     (IVIDENC1 InArgs *) &inArgs,
                    (IVIDENC1 OutArgs *)
                     &outArgs);
……
……

}
```

‖ Name

`process()` – basic encoding/decoding call

‖ Synopsis

```
XDAS_Int32 (*process)(IVIDENC1_Handle handle, XDM1_BufDesc
*inBufs, XDM1_BufDesc *outBufs, IVIDENC1_InArgs *inargs,
IVIDENC1_OutArgs *outargs);
```

‖ Arguments

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

XDM1_BufDesc *inBufs; /* algorithm input buffer descriptor
*/

XDM1_BufDesc *outBufs; /* algorithm output buffer
descriptor */

IVIDENC1_InArgs *inargs /* algorithm run-time input arguments
*/

IVIDENC1_OutArgs *outargs /* algorithm run-time 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.

‖ Post conditions

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

- If the process operation is successful, the return value from this operation is equal to `IALG_EOK`; otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.
- 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 section 3.4.

‖ 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()`

Appendix A

Revision History

This revision history highlights the changes made to SPRUEV1A codec specific user guide to make it SPRUEV1B

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

Section	Addition/Modification/Deletion
Section 2.7.3	Encoder Sample Base Param Setting: This section contains the values of IMP4VENC extended class variables for base param setting
Section 2.7.4	Encoder Sample Extended Param Setting: This section contains the values of IMP4VENC extended class variables for extended param setting