MPEG4/H263 Decoder on HDVCIP2 and Media Controller Based Platform

User's Guide

Literature Number: SPRUGQ8
November 2013

Products		**Applications**	
Audio	www.ti.com/audio	Automotive & Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications & Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers & Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energyapps
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video & Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Preface

Read This First

About This Manual

This document describes how to install and work with Texas Instruments' (TI) MPEG4 Advanced Simple Profile Decoder implementation on the HDVCIP2 . It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

TI's codec implementations based on the eXpress-DSP Digital Media (XDM) standard and IRES interface. XDM is an extension of the eXpress-DSP Algorithm Interface Standard (XDAIS). IRES are the universal resource manager to handle different types of resource requests by algorithms (codec). IRES keep the framework independent and agnostic of different resource requests by different algorithms

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 HDVCIP2 and Ducati Functional Simulator.

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

How to Use This Manual

This document includes the following chapters:

- **Chapter 1 - Introduction**, introduces 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.
- **Chapter 5 – Frequently Asked Questions,** Provides answers to few frequently asked questions related to using this decoder.

- **Chapter 6 - Debug Trace Usage,** Provides information on enabling decoder dump debug trace and collection procedure by application.
- **Appendix A- Picture Format,** Provides information on format of YUV buffers provide to Decoder.
- **Appendix B- Meta Data Support,** Provides information on writing the MB info data into application provided buffers.
- **Appendix C- Error Handling,** Provides information on handling the erroneous conditions while decoding.
- **Appendix D- Parse Header Support,** Provides information on parse header support for MPEG4 streams.
- **Appendix E- Support for display delay,** Provides information on configuration of decoder to achieve desired display delay.
- **Appendix F- Support for padding Type,** Provides information on configuration of decoder to support padding type for non standard resolution video clips (non multiple of 16 resolution)
- **Appendix G- Support for Dynamic change of resolution,** Provides information when given test stream is having dynamic change in resolution
- **Appendix H- Support for drop of frame ,** Provides info, when first frame of stream is missing from the sequence

Related Documentation from Texas Instruments

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

- *TMS320 DSP Algorithm Standard Rules and Guidelines* (literature number SPRU352) defines a set of requirements for DSP algorithms that, if followed, allow system integrators to quickly assemble production-quality systems from one or more such algorithms.
- *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360) describes all the APIs that are defined by the TMS320 DSP Algorithm Interoperability Standard (also known as XDAIS) specification.
- *Technical Overview of eXpressDSP - Compliant Algorithms for DSP Software Producers* (literature number SPRA579) describes how to make algorithms compliant with the TMS320 DSP Algorithm Standard which is part of TI's eXpressDSP technology initiative.
- *Using the TMS320 DSP Algorithm Standard in a Static DSP System* (literature number SPRA577) describes how an eXpressDSP-compliant algorithm may be used effectively in a static system with limited memory.
- *DMA Guide for eXpressDSP-Compliant Algorithm Producers and Consumers* (literature number SPRA445) describes the DMA architecture specified by the TMS320 DSP Algorithm Standard

(XDAIS). It also describes two sets of APIs used for accessing DMA resources: the IDMA2 abstract interface and the ACPY2 library.

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

The following documents describe TMS320 devices and related support tools:

- *Design and Implementation of an eXpressDSP-Compliant DMA Manager for C6X1X* (literature number SPRA789) describes a C6x1x-optimized (C6211, C6711) ACPY2 library implementation and DMA Resource Manager.
- *TMS320c64x+ Megamodule* (literature number SPRAA68) describes the enhancements made to the internal memory and describes the new features have been added to support the internal memory architecture's performance and protection.
- *TMS320C64x+ DSP Megamodule Reference Guide* (literature number SPRU871) describes the C64x+ megamodule peripherals.
- *TMS320C64x to TMS320C64x+ CPU Migration Guide* (literature number SPRAA84) describes migration from the Texas Instruments TMS320C64x™ digital signal processor (DSP) to the TMS320C64x+™ DSP.
- *TMS320C6000 Optimizing Compiler v 6.0 Beta User's Guide* (literature number SPRU187N) explains how to use compiler tools such as compiler, assembly optimizer, standalone simulator, library-build utility, and C++ name demangler.
- *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (literature number SPRU732) describes the CPU architecture, pipeline, instruction set, and interrupts of the C64x and C64x+ DSPs.
- *The Future of Digital Video White Paper* (literature number SPRY066)

Related Documentation

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

- *ISO/IEC 14496-10:2005 (E) Rec.- Information technology – Coding of audio-visual objects – H.264 (E) ITU-T Recommendation*

Abbreviations

The following abbreviations used in this document.

Table 1-1. List of Abbreviations

Abbreviation	Description
BIOS	TI’s simple RTOS for DSPs
CPB	Coded Picture Buffer

Abbreviation	Description
CSL	Chip Support Library
D1	720x576 resolutions in progressive scan
DCT	Discrete Cosine Transform
DMA	Direct Memory Access
DMAN	DMA Manager
DPB	Decoded Picture Buffer
EVM	Evaluation Module
GMC	Global Motion Compensation
HDTV	High Definition Television
IPCM	Intra-frame Pulse Code Modulation
IRES	Interface standard to request and receive handles to resources
IVA	Image Video Accelerator
IEC	International Electro technical Commission
ISO	International standard organization
MB	Macro Block
MMCO	Memory Management Control Operation
MPEG	Moving Pictures Experts Group
MV	Motion Vector
NTSC	National Television Standards Committee
RMAN	Resource Manager
RTOS	Real Time Operating System
UUID	Unregistered Unique Identifier
VGA	Video Graphics Array (640 x 480 resolution)
VOP	Video Object Plane
XDAIS	eXpressDSP Algorithm Interface Standard
XDM	eXpressDSP Digital Media
YUV	Color space in luminance and chrominance form

Text Conventions

The following conventions used in this document:

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

Product Support

When contacting TI for support on this codec, quote the product name (MPEG4 Advanced Simple Profile Decoder on HDVCIP2) and version number. The version number of the codec is included in the Title of the Release Notes that accompanies this codec.

Trademarks

Contents

Figures

This page is intentionally left blank

Tables

This page is intentionally left blank

Chapter 1

Introduction

This chapter introduces XDAIS and XDM. It also provides an overview of TI's implementation of the MPEG4 Advanced Simple Profile Decoder on the HDVCIP2 and Media Controller Based 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).

1.1.1 XDAIS Overview

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

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

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

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

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

1.1.2 XDM Overview

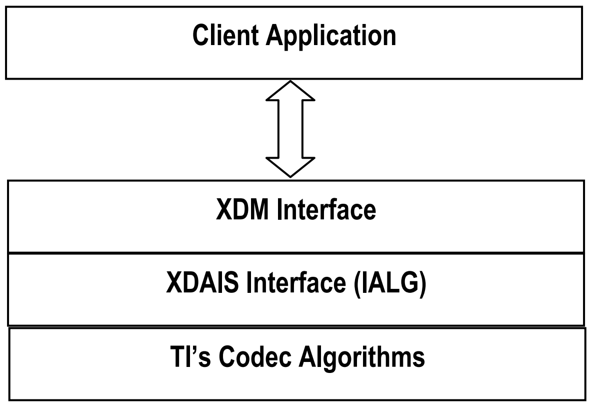
In the multimedia application space, you have the choice of integrating any codec into your multimedia system. For example, if you are building a video decoder system, you can use any of the available video decoders (such as MPEG4, H.263, or H.264) in your system. To enable easy integration with the client application, it is important that all codec’s with similar functionality use similar APIs. XDM was primarily defined as an extension to XDAIS to ensure uniformity across different classes of codec’s (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 codec's, XDM also standardizes the generic parameters that the client application must pass to these APIs. The client application can define additional implementation specific parameters using extended data structures.

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



As depicted in the figure, XDM is an extension to XDAIS and forms an interface between the client application and the codec component. XDM insulates the client application from component-level changes. Since TI's multimedia algorithms are XDM compliant, it provides you with the flexibility to use any TI algorithm without changing the client application code. For example, if you have developed a client application using an XDM-compliant MPEG4 video decoder, then you can easily replace MPEG4 with another XDM-compliant video decoder, say H.263, with minimal changes to the client application.

For more details, see *eXpressDSP Digital Media (XDM) Standard API Reference* (literature number SPRUEC8).

1.1.3 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 define 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 requested. The framework calls the IRES interface functions, in addition to the IALG functions, to perform IRES resource initialization, activation, and deactivation.

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

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

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

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

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

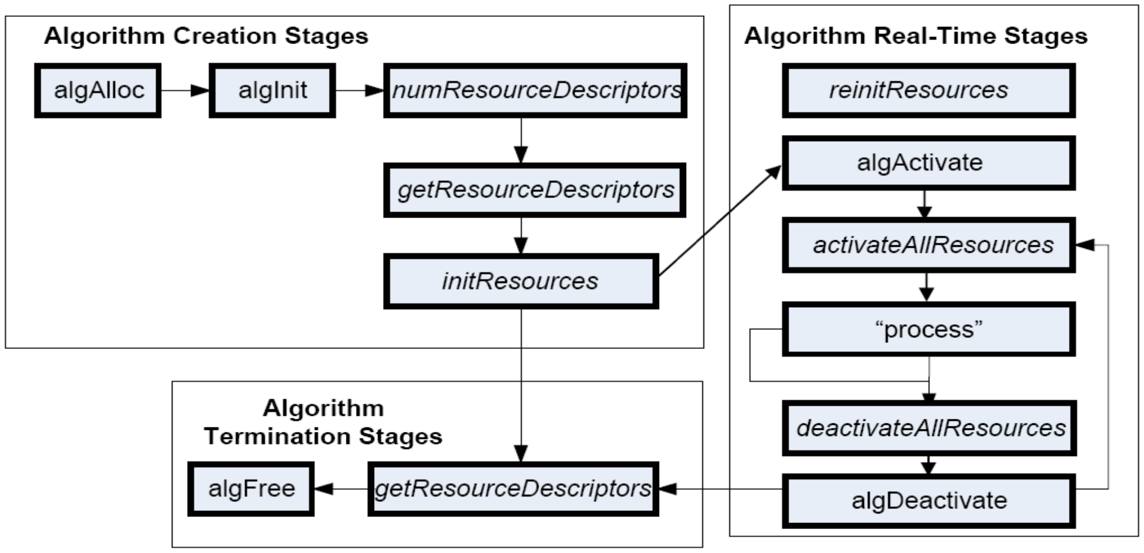


Figure 1-1. IRES Interface Definition and Function Calling Sequence.

1.2 Overview of MPEG4 Advanced Simple Profile Decoder

MPEG4 is a popular video coding algorithm enabling high quality multimedia services on a limited bandwidth network. MPEG4 standard defines several profiles and levels that specify restrictions on the bit-stream and hence limits the capabilities needed to decode the bit-streams. Each profile specifies a subset of algorithmic features that limits all decoders conforming to that profile may support. Each level specifies a set of limits on the values that may taken by the syntax elements in that profile. There are 19 profiles defined for MPEG4 video coding standard.

Some important MPEG4 profiles and their special features:

- Simple Profile:
  - I-VOP (Intra-coded rectangular VOP, progressive video format)
  - P-VOP (Inter-coded rectangular VOP, progressive video format)
  - Short header (mode for compatibility with H.263 Codec’s)
  - Compression efficiency tools (four motion vectors per macro block, unrestricted motion vectors, Intra prediction).
  - Transmission efficiency tools(video packets, Data Partitioning, Reversible Variable Length Codes)
- Advanced Simple Profile:
  - I-VOP (Intra-coded rectangular VOP, progressive video format)
  - P-VOP (Inter-coded rectangular VOP, progressive video format)
  - B-VOP (bi-directionally predicted Inter-coded VOP)
  - Quarter-pixel motion compensation.
  - Global motion compensation.
  - Two quantization modes
  - Interlace (tools for coding interlaced video sequences).
  - Short header (mode for compatibility with H.263 Codec’s)
  - Compression efficiency tools (four motion vectors per macro block, unrestricted motion vectors, Intra prediction).
  - Transmission efficiency tools(video packets, Data Partitioning, Reversible Variable Length Codes)

The input to the decoder can be MPEG-4, short header specific encoded bit-stream in the byte-stream syntax. Each byte-stream contains sequence of syntax structure like, VOS (Video object sequence), VO (Visual object), VOL (Video Object layer), VO(Video object),GOV(Group of VOP) and VOP(Video object plane. All the above syntax structures have their own length defined by the standard and also some of them may or may not be present in encoded streams. Each VOP will be containing Intra, Inter, MV data for decoding and reconstructing the frame.

- **Intra coded data**: - Spatial prediction mode (AC DC Co-efficient prediction from neighboring intra MB/BLOCKS) and prediction error data that subjected to DCT and later quantized.

- **Inter coded data**: - Motion information (Motion vector) and residual error data (differential data between two frames) that is subjected to DCT and later quantized.

The decoder re-constructs an Intra frame by spatial intra-prediction specified by the mode and by adding the prediction error. In case of inter coding, the decoder reconstructs the frame by adding the residual error data to the previously decoded picture, at the location specified by the motion information (Motion vector). The output of the decoder is a YUV sequence, which is of 4:2:0 semi planar format(Y is a single plane and the Chroma data – Cb and Cr are interleaved to form the other plane).

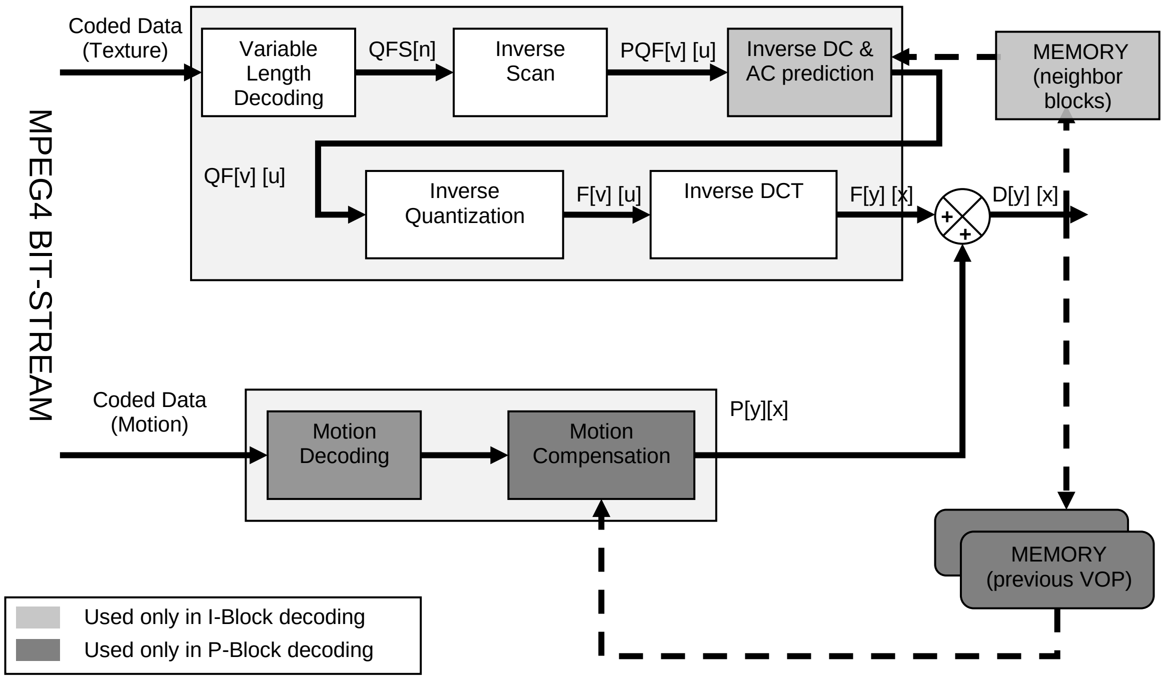


Figure 1-2. Flow diagram of the MPEG4 Advanced Simple Profile Decoder.

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

1.3 Supported Services and Features

This user guide accompanies TI's implementation of MPEG4 Decoder on the HDVCIP2 and Media Controller Based platform.

This version of the codec has the following supported features:

- MPEG4 Simple profile levels 0,1,2,3,4A,5 and 6 are supported
- MPEG4 Advanced Simple Profile levels 0,1,2,3,4,5 and 6 are supported
- Support for H.263 profile 0 and 3
- Supports H.263 Annexs I,J,K and T only
- Supports Progressive, Interlace picture type decoding
- Supports intra-prediction and inter-prediction modes
- Supports frame based decoding
- Frame with minimum resolution of 64x64 to maximum 2048x2048 pixel supported
- Frame with width/height non-multiple of two decoding is supported.
- Supports YUV420 semi-planar Chroma format
- Support for graceful exit under error conditions
- Error concelement for errorneuos streams is supported
- Parse header functionality supported
- Codec to provide MB info to application as part of Meta data infomation is supported when transecode mode is on and this is optional
- Performance measured for 1920x1080 picture resolution for progressive as well interlace format in normal conditions (without error scenario)
- Configurable display delay in case of low delay application supported
- eXpressDSP Digital Media (XDM IVIDDEC3) compliance
- Supports booting of HDVCIP2 and power optimization techniques
- Integrated with codec engine using FC version 3.20.00.22
- Integrated with HDVICP2.0 library version 01.00.00.23
- Provides library that can be used with RTSC as well as non-RTSC environment for system integration
- Ability to plug in any multimedia frameworks (e.g. Codec engine, OpenMax, GStreamer etc)
- Independent of any OS (DSP/BIOS, Linux, windowCE, syybian etc.)

- Supports multi-channel functionality
- Supports optional loop filter post-processing

This version of the decoder does not support the following features as per the Advanced Simple Profile feature set:

- Support for Global Motion Compensation
- Support for Sub-frame level data synchronization API's at both Input and Output

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

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

2.1.1 Hardware

This codec is been tested on the HDVCIP2 and Media Controller based OMAP4 and DM816x DDR2 EVM REV-B hardware platforms.

2.1.2 Software

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

- **Development Environment:** This codec is developed using Code Composer Studio (Code Composer Studio v4) version 4.2.0.09000.

  http://softwaredl.ti.com/dsps/dsps_registered_sw/sdo_ccstudio/CCSv4/Prereleases/setup_CCS_4.2.0.09000.zip

- **Code Generation Tools:** This codec is compiled, assembled, archived, and linked using the code generation tools version 4.5.1 for HDVICP2 processor and 5.0.3 for Media Controller Processor.

  All though CG tools are part of Code Composer Studio v4 installation, it is recommended that you re-install CG tools by downloading from the following link.

  https://www-a.ti.com/downloads/sds_support/CodeGenerationTools.htm

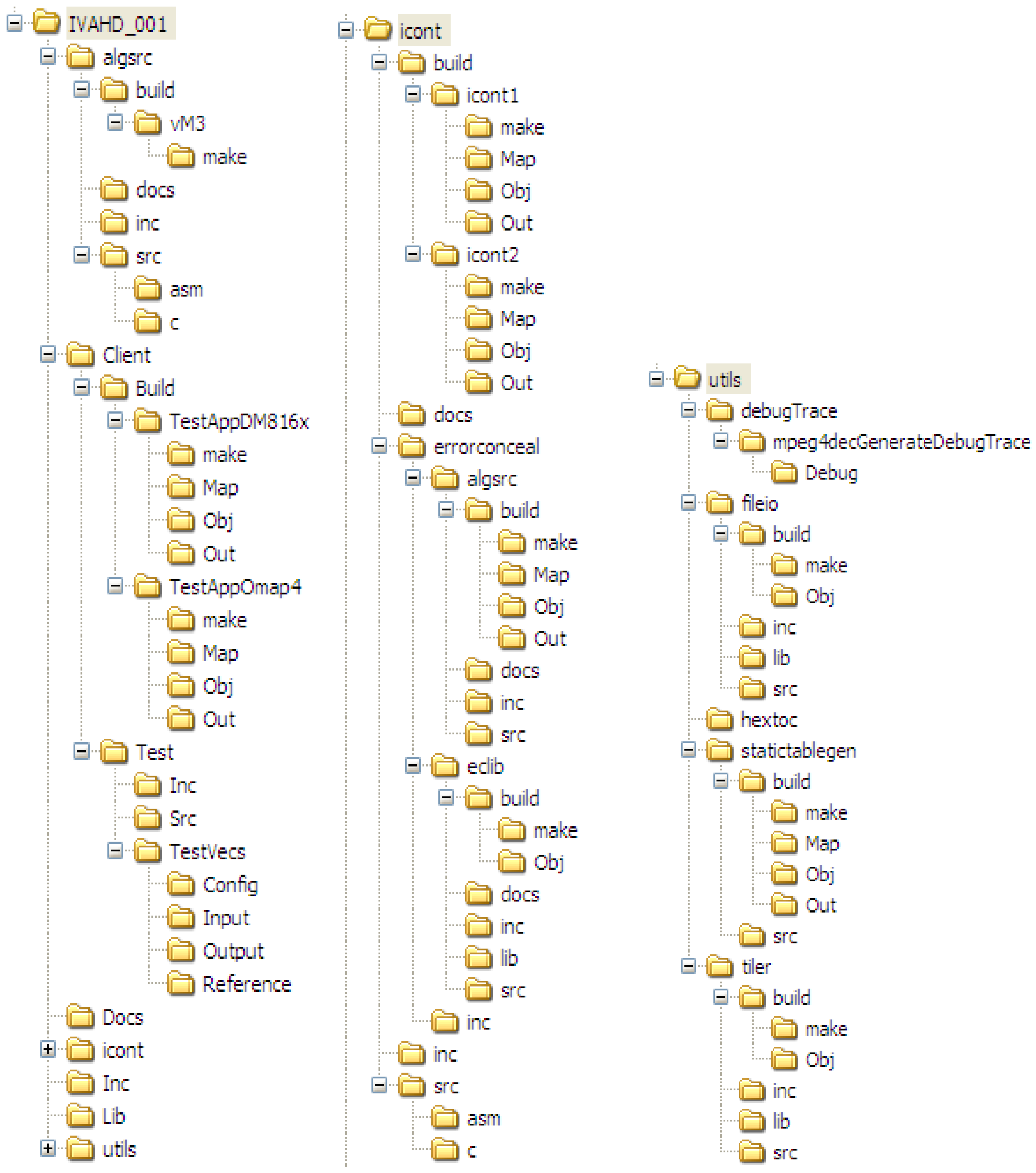
  Platform Simulator: This codec is developed using Netra/OMAP4 simulator with CSP version 0.7.1.This release can be obtained by software updates on code composer studio v4. Make sure that following site is listed as part of updates sites to visit.

  http://softwaredl.ti.com/dsps/dsps_public_sw/sdo_ccstudio/CCSv4/Updates/NETRA/site.xml

2.2 Installing the Component

The codec component is released as a compressed archive. To install the codec, extract the contents of the zip file onto your local hard disk. The zip file extraction creates a directory called 500.V.MPEG4.D.ASP.IVAHD.01.00 under which under which the directory named IVAHD_001 is created:

The sub directory structures for IVAHD_001 are depicted in Figure 2- 1



```
IVAHD_001
    algsrc
        build
            vM3
                make
        docs
        inc
        src
            asm
            c
    Client
        Build
            TestAppDM816x
                make
                Map
                Obj
                Out
            TestAppOmap4
                make
                Map
                Obj
                Out
        Test
            Inc
            Src
            TestVecs
                Config
                Input
                Output
                Reference
    Docs
    icont
    Inc
    Lib
    utils

icont
    build
        icont1
            make
            Map
            Obj
            Out
        icont2
            make
            Map
            Obj
            Out
    docs
    errorconceal
        algsrc
            build
                make
                Map
                Obj
                Out
            docs
            inc
            src
        eclib
            build
                make
                Obj
            docs
            inc
            lib
            src
        inc
    inc
    src
        asm
        c

utils
    debugTrace
        mpeg4decGenerateDebugTrace
            Debug
    fileio
        build
            make
            Obj
        inc
        lib
        src
    hextoc
    statictablegen
        build
            make
            Map
            Obj
            Out
        src
    tiler
        build
            make
            Obj
        inc
        lib
        src
```

Figure 2- 1.Component Directory Structure

Table 2-1 provides a description of the sub-directories created in IVAHD_001directory.

Table 2-1. Component Directories

Sub-Directory	**Description**

Sub-Directory	**Description**
\algsrc\build\vM3\Obj	Contains intermediate Object files generated for Media Controller (host) project
\algsrc\docs	Contains documents specific to the Media Controller (host) project
\algsrc\inc	Contains header files needed by the Media Controller (host) project and some interface files which are shared between HDVCIP2 and Media Controller
\algsrc\src\asm	Contains assembly files needed by Media Controller (host) project
\algsrc\src\c	Contains source files needed by Media Controller (host) project
\Client\Build\ TestAppDeviceName	Contains the CCSv4 project files. The name of this directory will not be same as exactly mentioned here. Instead of Device Name string, actual name of Device will be present
\Client \Build\ TestAppDeviceName \Map	Contains the memory map file, generated on compilation of the code.
\Client \Build\ TestAppDeviceName \Obj	Contains the intermediate .asm and/or .obj file generated on compilation of the code
\Client \Build\ TestAppDeviceName \Out	Contains the final application executable (.out) file generated by the sample test application.
\ Client \Test\Src	Contains application source files
\ Client \Test\Inc	Contains application header files
\ Client \Test\TestVecs\Config	Contains sample configuration files for mpeg4 decoder
\ Client \Test\TestVecs\Input	Contains input test vectors
\ Client \Test\TestVecs\Output	Contains output generated by the codec. It is empty directory as part of release
\ Client \Test\TestVecs\Reference	Contains read-only reference output to be used for cross-checking against codec output

Sub-Directory	**Description**
\docs	Contains user guide, and data sheet.
\icont\build\icont1\Map	Contains the generated map file related to icont1 project
\icont\build\icont1\Obj	Contains the generated object files related to icont1 project
\icont\build\icont1\Out	Contains the generated executable files related to icont1 project
\icont\build\icont2\Map	Contains the generated map file related to icont2 project
\icont\build\icont2\Obj	Contains the generated object files related to icont2 project
\icont\build\icont2\Out	Contains the generated executable files related to icont2 project
\icont\docs	Contains the iCONT module specific documents
\icont\inc	Contains the iCONT module specific header files
\icont\src\asm	Contains assembly files needed by the iCONT1 and 2 projects
\icont\src\c	Contains source files needed by the iCONT1 and 2 projects
\icont\errorconceal\algsrc\ build \Map	Contains the generated map file related to errorconcealment
\icont\errorconceal\algsrc\build\Obj	Contains the generated object files related to errorconcealment
\icont\errorconceal\algsrc\build\Out	Contains the generated executable files related to errorconcealment
\icont\errorconceal\algsrc\build\make	Contains the generated executable files related errorconcealment
\icont\errorconceal\algsrc\docs	Contains the errorconcealment module specific documents
\icont\errorconceal\algsrc\inc	Contains the errorconcealment module specific header files
\icont\errorconceal\algsrc\src	Contains source files needed by the errorconcealment.
\icont\errorconceal\eclib\ build \Map	Contains the generated map file related to errorconcealment eclib
\icont\errorconceal\eclib\build\Obj	Contains the generated object files related to errorconcealment eclib
\icont\errorconceal\eclib\build\Out	Contains the generated executable files related to errorconcealment eclib
\icont\errorconceal\eclib\build\make	Contains the generated executable files related errorconcealment eclib

Sub-Directory	Description
\icont\errorconceal\eclib\docs	Contains the errorconcealment eclib module specific documents
\icont\errorconceal\eclib\inc	Contains the errorconcealment eclib module specific header files
\icont\errorconceal\eclib\src	Contains source files needed by the errorconcealment eclib.
\icont\errorconceal\eclib\lib	Contains library generated by the errorconcealment eclib.
\icont\errorconceal\inc	Contains header files.
\Inc	Contains XDM related header files, which allow interface to the codec library.
\Lib\	Contains the library file named as mpeg4vdec_ti_host.lib for decoding the compressed video data
\utils	Contains the utility file(s) required by mpeg4 decoder
\utils\debugTrace	Contains visual c project to generate debug trace along with executable to generate the debug trace for decoder.
\utils\fileio	Contains makefile based project and required source and header files to support various file read and write lib options along with this also present common lib fileio.lib.
\utils\statictablegen	Contains the project and source files to generate the static tables/commands offline.
\utils\statictablegen\build	Contains the project to generate the static tables/commands offline.
\utils\statictablegen\build\Map	Contains the generated map file related to static tables/commands Preparation project.
\utils\statictablegen\build\Obj	Contains the generated obj file related to static tables/commands preparation project.
\utils\statictablegen\build\Out	Contains the generated executable file related to static tables/commands Preparation project.
\utils\statictablegen\src	Contains the source file to generate the static tables/commands offline.
\utils\Hextoc	Contains the source file to generate executable for hex to c conversion.
\utils\tiler	Contains makefile based project and required source and header files to support generation of common tiler memory related lib.

2.3 Before Building the Sample Test Application

This codec is accompanied by a sample test application. To run the sample test application, you need TI Framework Components (FC) and HDVICP2 library.

This version of the codec has been validated Framework Component (FC) version 3.20.00.22.

This version of the codec has been validated HDVICP2 library version 01.00.00.23 and HDVICP2.0 CSP Version 00.05.02.00

Set the system environment variable TI_DIR to the CCSv4 installation path. Example: TI_DIR = <CCSv4 Installation Dir>\ccsv4.

Add gmake (GNU Make version 3.78.1) utility folder path (for example, “C:\CCStudioV4.0\ccsv4\utils\gmake”) at the beginning of the PATH environment variable.

Install CG Tools version 4.5.1 for ARM (TMS470) at the following location in your system: <CCSv4.2_InstallFolder>\ccsv4\tools\compiler\tms470. CGTools 4.5.1 & 5.0.3 can be downloaded from

https://www-a.ti.com/downloads/sds_support/CodeGenerationTools.htm

Please note that CG Tools 4.5.1 is installed at the location mentioned above along with the CCS v4.2 installation by default. However, as some problems have been reported about this, we recommend that you install CG Tools 4.5.1 again with the installer obtained from the above link. CG tool 5.0.3 can be installed at any path.

Set environment variable CG_TOOL_DIR to <cgtools4.5.1_install_dir>\

Set environment variable CG_TOOL_DIR_M3 to <cgtools5.0.3_install_dir>\

The version of the XDC tools required is 3.20.04.68.

2.3.1 Installing Framework Component (FC)

You can download FC from the following website:

http://softwaredl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/fc/3_20_00_22/index_FDS.html

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

<Install directory>\CCStudio4.0

Set a system environment variable named FC_INSTALL_DIR pointing to <install directory>\CCStudio4.0\<fc_directory>

The test application uses the following IRES and XDM files:

- HDVICP related ires header files, these are available in the <install directory>\CCStudio4.0\<fc_directory>\packages \ti\sdo\fc\ires\hdvicp directory.

- Tiled memory related header file, these are available in the <install directory>\CStudio4.0\<fc_directory>\fctools\packages \ti\sdo\fc\ires\tiledmemory directory.
- XDM related header files, these are available in the <install directory>\CCStudio4.0\<fc_directory>\fctools\packages \ti\xdais directory.
- Memutils file for memory address translation, these are available in the <install directory>\CStudio4.0\<fc_directory>\ packages\ti\sdo\fc\memutils directory

2.3.2 Installing XDC Tools

XDC Tools is required to build the test application. The test application uses the standard files like <std.h> from XDC tools. This decoder has been validated with XDC version 3.20.04.68. The XDC tools can be downloaded and installed from the following URL:

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/rtsc/3_20_04_68/index_FDS.html

Also, ensure that the environment variable XDCROOT is set to the XDC installation directory.

2.3.3 Installing HDVICP2 and CSP library

The HDVICP2 library should be available in the same place as the codec package.

Set a system environment variable named HDVICP2_INSTALL_DIR pointing to <hdvicp2_directory>\hdvicp20

The test application uses the HDVICP20 library file (ivahd_ti_api_vM3.lib) from <hdvicp2_directory>\hdvicp20\lib directory

Set a system environment variable named CSP_INSTALL_DIR pointing to <csp_directory>\csp

2.4 Building and Running the Sample Test Application

2.4.1 Building the Sample Test Application

MPEG4 decoder on HDVICP2 and Media Controller based platform has below projects

Project	Make file Path	Output files
Icont 1	\icont\build\icont1\make\	\icont\build\icont1\out\ mpeg4vdec_ti_icont1.out \algsrc\inc\ mpeg4vdec_ti_icont1_code.h \algsrc\inc\ mpeg4vdec_ti_icont1_code_debugtracelevel1.h

		\algsrc\inc\ mpeg4vdec_ti_icont1_code_debugtracelevel2.h
Icont 2	\icont\build\icont2\make\	\icont\build\icont2\out\ mpeg4vdec_ti_icont2.out \algsrc\inc\ mpeg4vdec_ti_icont2_code.h \algsrc\inc\ mpeg4vdec_ti_icont2_code_debugtracelevel1.h \algsrc\inc\ mpeg4vdec_ti_icont2_code_debugtracelevel2.h
vM3	\algsrc\build\vM3\make\	\lib\ mpeg4vdec_ti_host.lib
Test Application	\client\build\<TestAppDeviceName>\make\	\client\build\TestApp<DeviceName>\out \ mpeg4vdec_ti_hosttestapp.out

Run the <release_package>\make.bat with "all' as an argument to build all the projects.

Ex: `make.bat all`

This batch file will build all the projects in the above mentioned order and generate the output files as given in the table.

Below command can be used for cleaning all the projects

```
make.bat clean
```

Individual make files for each project can be built using the below commands

```
gmake –k –s clean

gmake –k –s deps

gmake –k –s all
```

Note: To build sample test application to run on OMAP4 simulator, macro "HOSTCORTEXM3_SIMULATOR" need to be defined in the test application make file placed at \client\build\<TestAppDeviceName>\make\

2.4.2 Running the Sample Test Application on OMAP4 ES1.0

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 on OMAP4 ES1.0, follow these steps:

1) Start Code Composer Studio v4 and set up the target configuration for OMAP4 ES1.0 Emulator.

2) Select the Debug perspective in the workbench. Launch OMAP4 ES1.0 Emulator in CCSv4 (**View > Target Configurations > %OMAP4 Emulator%**).

3) Select CortexA9_0 device, right click and choose "Connect Target" and wait for emulator to connect to CortexA9 and execute the GEL file (omap4430 startup sequence).

4) Select Cortex_M3_0 device, right click and choose "Connect Target" and wait for emulator to connect to CortexM3.

5) Select Cortex_M3_0 device and **Target > Load Program**, browse to the \client\build\TestApp<DeviceName>\out\ sub-directory, select the codec executable “mpeg4vdec_ti_hosttestapp.out” and load it in preparation for execution.

6) Select **Target > Run** to execute the application for Cortex_M3_0 device.

7) Test application will take input streams from \client\test\testvecs\input\ directory and generates outputs in \client\test\testvecs\output\ directory.

Note:

Order of connecting to the devices is important and it should be as mentioned in above steps.

2.4.3 Running the Sample Test Application on DM816x DDR2 EVM REV-B

To run the sample test application on DM816x DDR2 EVM, follow these steps:

1) Select the Debug perspective in the workbench. Launch DM816x DDR2 EVM Emulator configuration in CCSv4.

2) Select CortxA8 device, right click and choose “Connect Target” and wait for emulator to connect to CortexA8.

3) Select **Tools > GEL Files**. This opens up the GEL Files window. Right click and select “**Load GEL…**”. Load the GEL file named DM816x_Rev_A_DDR2_EVM.gel.

4) Select **Scripts > NETRA Omx Init > OmxInit**. The script runs. Wait till you see “Omx Initialization completed”.

5) Select Cortex_M3_RTOS_0 device, right click and choose “Connect Target” and wait for emulator to connect to CortexM3.

6) Select Cortex_M3_RTOS_0 device and **Target > Load Program**, browse to \client\build\TestAppDM816x\out\ sub-directory, select the codec executable “mpeg4vdec_ti_hosttestapp.out” and load it in preparation for execution.

7) Select **Target > Run** to execute the application for Cortex_M3_RTOS_0 device.

8) Test application will take input streams from \client\test\testvecs\input\ directory and generates outputs in \client\test\testvecs\output\ directory.

Note:

Order of connecting to the devices is important and it should be as mentioned in above steps.

2.5 Configuration Files

This codec is shipped along with:

- ❑ Generic configuration file (Testvecs.cfg) – specifies input configuration file.
- ❑ Decoder configuration file (airshow_p176x144_nv12.cfg) – specifies the configuration parameters used by the test application to configure the Decoder for given test stream.

2.5.1 Generic Configuration File

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

A sample Testvecs.cfg file is as shown:

```
..\..\..\Test\TestVecs\Config\airshow_p176x144_nv12.cfg
```

Above is describing the input test stream configuration file path. Input test stream configuration file will be having all information related to configuring the decoder to decode given test stream.

2.5.2 Decoder Configuration File

The decoder configuration file, `airshow_p176x144_nv12.cfg` contains the configuration parameters required for the decoder for the mpeg4 stream. Configuration file is available in the \Client\Test\TestVecs\Config sub-directory.

A sample `airshow_p176x144_nv12.cfg` file is as shown:

```
###################################################################
# Input and Output
###################################################################
inputBitStream       =
"..\..\..\Test\TestVecs\Input\airshow_p176x144_nv12.m4v"
outputYUV            =
"..\..\..\Test\TestVecs\Output\airshow_p176x144_nv12.yuv"
referenceYUV         =
"..\..\..\Test\TestVecs\Reference\airshow_p176x144_nv12.yuv"
frameSizeFile        =
"..\..\..\Test\TestVecs\Input\airshow_p176x144_nv12.txt"
TestCompliance       = 0         # 0->Dump Mode ,1->[Compare Mode Not
supported]
###################################################################
# Create Time Parameters
###################################################################
maxHeight            = 144      # Max Image height in Pels
maxWidth             = 176      # Max Image width in Pels
maxFrameRate         = 30      # 30 -> Frame rate in fps
```

```
maxBitRate           = 10485760      # Maximum Bit rate in Bytes
dataEndianness       = 1      # 1 -> 8-bit Big Endian stream.
forceChromaFormat    = 9      # 9 -> XDM_YUV_420SP
operatingMode        = 0      # 0 -> Decode Mode, 2->Transcode
displayDelay         = -1      # 0 -> No delay (Decode order)
inputDataMode        = 3      # 3->Frame Mode, 0,1 -> Sub-Frame (DataSync)
Mode
outputDataMode       = 3      # 3->Frame Mode, 2 -> Sub-Frame (DataSync)
Mode
numInputDataUnits    = 0      # 0 -> Non-DS mode. Non-Zero positive for DS
mode
numOutputDataUnits   = 0      # 0 -> Non-DS mode. Non-Zero positive for DS
mode
errorInfoMode        = 0      # 0 -> Error Info off
displayBufsMode      = 2      # 1 -> Embedded, 2 - Pointer to struct
metadataType_0       = -1      # -1->No Metadata, 0- MB Info
metadataType_1       = -1      # -1->No Metadata
metadataType_2       = -1      # -1->No Metadata
outloopDeBlocking    = 0      # 0 -> Disable optional filtring, 1-> enable
                                2->Enhanced Deblocking
enhancedDeBlockingQp = 31      # Range 1 to 31
errorConcealmentEnable = 1      # 0 -> Disable EC, 1-> enable EC
sorensonSparkStream  = 0      # 0 -> Disable Sorenson spark decoding,1->
enable
debugTraceLevel      = 0      # 0 -> Disable debug trace, 1,2-> Enable
lastNFramesToLog     = 0      # 0 -> Default, 1-9 supported
paddingType          = 0      # 0 -> Default(divx type of padding), 1->
mpeg4 padding
decodeOnlyIntraFrames = 0     # 0 -> Disable decoding of only I frames
feature, 1-> Enable the feature
Rsvd2                = 0      # 0 -> Default, reserved one for future use
###################################################################
# Dynamic Parameters
###################################################################
decodeHeader         = 0         # 0 -> Disable decode Header mode
displayWidth         = 0         # 0->Default, otherwise Positive value
newFrameFlag         = 1         # 1 -> True, 0-> false
lateAcquireArg       = 0         # 0->Default
DynRsvd0             = 0      # 0 -> Default, reserved one for future use
DynRsvd1             = 0      # 0 -> Default, reserved one for future use
DynRsvd2             = 0      # 0 -> Default, reserved one for future use
###################################################################
# Application Control Parameters
###################################################################
MbInfoWriteMode      = 0         # 0->disable mbinfo dump 1->Enable mbinfo
dump
TilerEnable          = 0         # 0 -> Disable, 1->Enable TILER
ChromaTilerMode      = 0         # 0 -> 16-Bit mode, 1->8-Bit Mode
BitStreamMode        = 0         # 0 -> Buffer Mode, 1->Frame size Mode
ReadHeaderData       = 0         # 0 -> default, 1->Header data given
separaty then residual
NumFramesToDecode    = 8000         # 8000 -> Default
parBoundCheck        = 0         # Parameter Boundary check: 0 -> Disable,
1-> Enable
parExpectedStatus    = 0         # Expected Status during Param Boundary
check. 0->Pass, -1 -> Fail
exitLevel            = 0         # 1->Create Time, 2->XDM control time
xdmReset             = 0         # 0->Disable XDM reset use, 1->Enable XDM
reset use
DumpFrom             = 0         # 0 -> Default, frame number to dump from
CRCEnable            = 0         # CRC check: 0 -> Disable, 1->Enable
ProfileEnable        = 0         # Frame level Profiling: 0 -> Disable, 1-
>Enable
```

```
BaseClassOnly        = 0         # 0 -> Use Extended classes, 1->Use Base
classes Only
ivahdID              = 0         # 0-> Default. Supports 1 & 2 for Netra
AppRsvd0             = 0         # 0 -> Default, reserved one for future use
AppRsvd1             = 0         # 0 -> Default, reserved one for future use
AppRsvd2             = 0         # 0 -> Default, reserved one for future use
```

Note:

Chroma Format supported in this codec is 420 semi-planar. That is, the chroma planes (Cb and Cr) are Interleaved.

2.6 Standards Conformance and User-Defined Inputs

To check the conformance of the codec for the default input file shipped along with the codec, follow the steps as described in Section 2.4. To check the conformance of the codec for other input files of your choice, follow these steps:

- ❑ Copy the input files to the \Client\Test\TestVecs\Inputs sub-directory
- ❑ Copy the reference files to the \Client\Test\TestVecs\Reference subdirectory.
- ❑ Edit the configuration file, TestVecs.cfg available in the \Client\Test\TestVecs\Config sub-directory. For details on the format of the TestVecs.cfg file, see Section 2.5.1.
- ❑ Prepare the *.cfg with given test stream to configured the decoder to decode the given stream.For details on the format of the decoder configuration file, see Section 2.5.12. or refer the file `airshow_p176x144_nv12.cfg` present in \Client\Test\TestVecs\Config directory.

2.7 Uninstalling the Component

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

This page is intentionally left blank

Chapter 3

Sample Usage

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

Topic **Page**

3.1 Overview of the Test Application

The test application exercises the `IVIDDEC3` base class of the MPEG4 Decoder library. The main test application files are mpeg4vdec_ti_hosttestapp.c and mpeg4vdec_rman_config.c. These files are available in the \Client\Test\Src directory.

Figure 3-1 depicts the sequence of APIs exercised in the sample test application. Currently, the test application does not use RMAN resource manager. However, all the resource allocations happen through IRES interfaces.

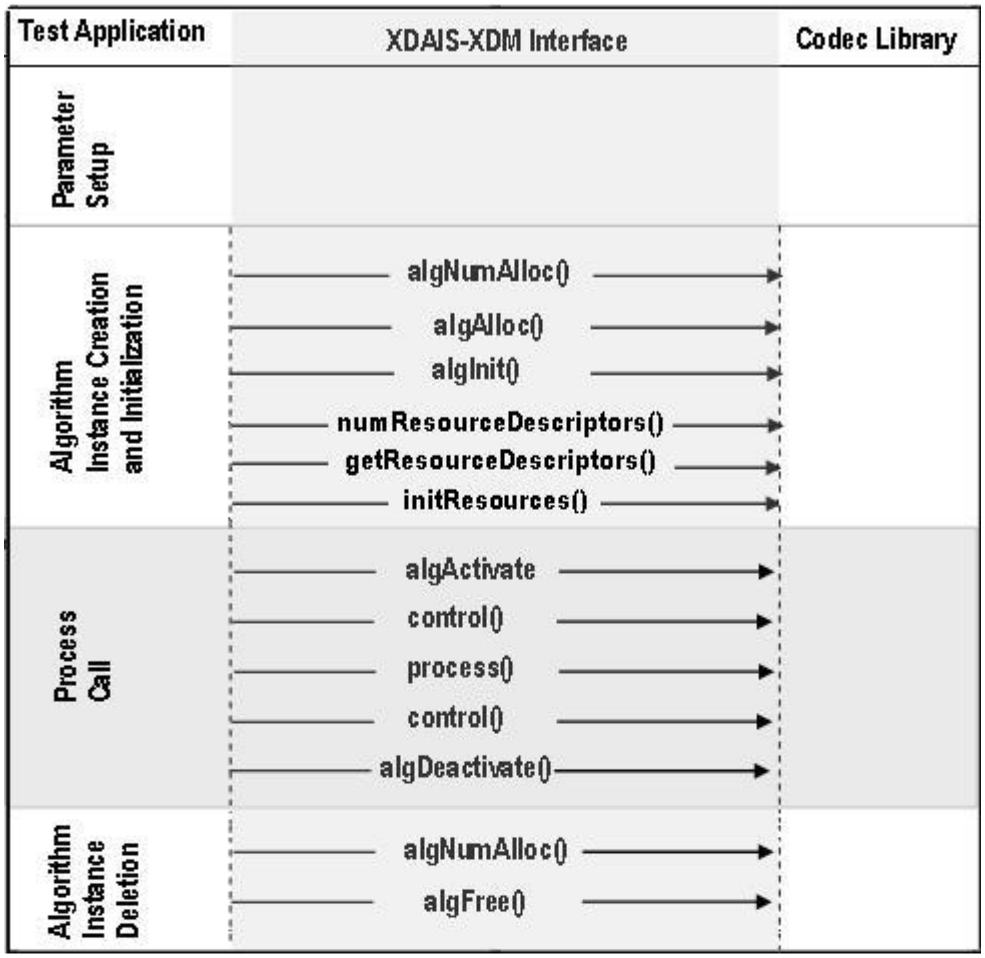


Figure 3-1. Test Application Sample Implementation

The test application is divided into four logical blocks:

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

3.1.1 Parameter Setup

Each codec component requires various codec configuration parameters to be set at initialization. For example, a video codec requires parameters such as video height, video width, and so on. The test application obtains the required parameters from the Decoder configuration files.

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

1) Opens the generic configuration file, Testvecs.cfg and reads the compliance checking parameter, Decoder configuration file name (Testparams.cfg), input file name, and output/reference file name.

2) Opens the Decoder configuration file, (Testparams.cfg) and reads the various configuration parameters required for the algorithm.

   For more details on the configuration files, see Section 2.5.

3) Sets the `IVIDDEC3_Params` structure based on the values it reads from the Testparams.cfg file.

4) Initializes the various DMAN3 parameters.

5) 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.

3.1.2 Algorithm Instance Creation and Initialization

In this logical block, the test application accepts the various initialization parameters and returns an algorithm instance pointer. The following APIs are called in sequence:

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

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

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

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

Note:

- Decoder requests only one memory buffer through algNumAlloc. This buffer is for the algorithm handle.
- Other memory buffer requirements are done through IRES interfaces.

After successful creation of the algorithm instance, the test application does HDVICP Resource and memory buffer allocation for the algorithm. Currently, RMAN resource manager is not used. However, all the resource allocations happen through IRES interfaces:

- `numResourceDescriptors()` - To understand the number of resources (HDVICP and buffers) needed by algorithm.
- `getResourceDescriptors()` - To get the attributes of the resources.
- `initResources()` - After resources are created, application gives the resources to algorithm through this API.

3.1.3 Process Call

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

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

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

3) Implements the process call based on the non-blocking mode of operation explained in step 4. The behavior of the algorithm can controlled using various dynamic parameters (see Section 4.2.1.8). The inputs to the `process()`functions are input and output buffer descriptors, pointer to the `IVIDDEC3_InArgs` and `IVIDDEC3_OutArgs` structures.

4) On the call to the `process()` function for encoding/decoding a single frame of data, the software triggers the start of encode/decode. After triggering the start of the encode/decode frame, the video task can be put to SEM-pend state using semaphores. On receipt of interrupt signal at the end of frame encode/decode, the application releases the semaphore and resume the video task, which does any bookkeeping operations by the codec and updates the output parameter of `IVIDDEC3_OutArgs` structure.

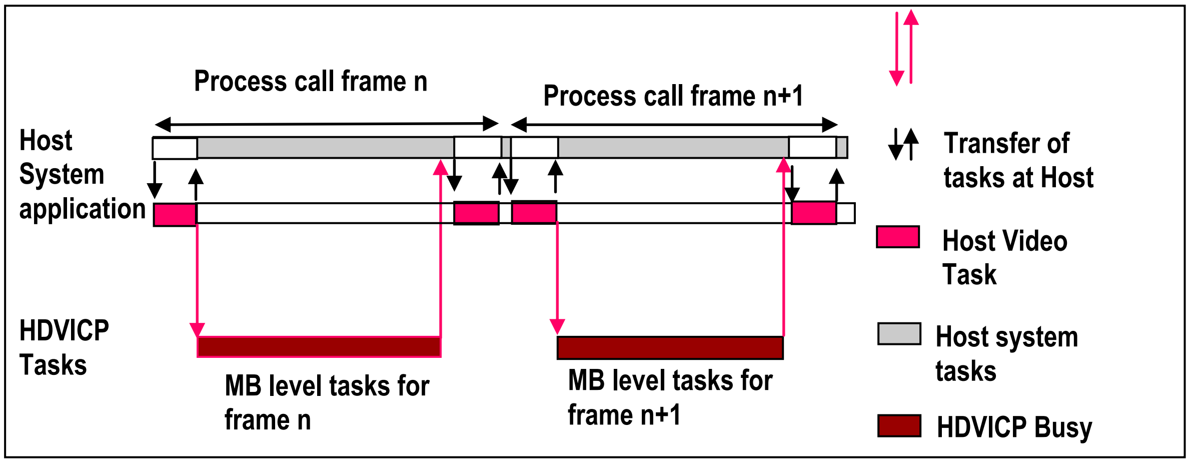


Figure 3-2. Process call with Host release.

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

- `algActivate()` - To activate the algorithm instance.
- `control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the six available control commands.
- `process()` - To call the Decoder with appropriate input/output buffer and arguments information.
- `control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the six available control commands.
- `algDeactivate()` - To deactivate the algorithm instance.

The do-while loop encapsulates picture level `process()` call and updates the input buffer pointer every time before the next call. The do-while loop breaks off either when an error condition occurs or when the input buffer exhausts. It also protects the `process()` call from file operations by placing appropriate calls for cache operations. The test application does a cache invalidate for the valid input buffers before `process()` and a cache write back invalidate for output buffers after a `control()` call with `GET_STATUS` command.

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

3.1.4 Algorithm Instance Deletion

Once decoding/encoding is complete, the test application frees the memory resources and deletes the current algorithm instance. The following APIs called in sequence:

1) `numResourceDescriptors()` - Get the number of resources and free them. If the application needs handles to the resources, it can call `getResourceDescriptors()`.

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

3) `algFree()` - To query the algorithm for memory, to free when removing an instance.

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

3.2 Frame Buffer Management by Application

3.2.1 Frame Buffer Input and Output

With the new XDM, decoder does not ask for frame buffer at the time of `alg_create()`. It uses buffer from `XDM2_BufDesc *OutBufs`, which it reads during each decode process call. Hence, there is no distinction between DPB and display buffers. The framework needs to ensure that it does not overwrite the buffers that are locked by the codec.

```
mp4VDEC_create();
mp4VDEC_control(XDM_GETBUFINFO); /* Returns default 1080p
HD size */
do{
mp4VDEC_decode(); //call the decode API
mp4VDEC_control(XDM_GETBUFINFO); /* updates the memory
required as per the size parsed in stream header */
}
while(all frames)
```

Note:

- Application can take the information retured by the control function with the `XDM_GETBUFINFO` command and change the size of the buffer passed in the next process call.
- Application can re-use the extra buffer space of the 1st frame, if the control call returns buffer that is of small size than that was provided.

The frame pointer given by the application and that returned by the algorithm may be different. `BufferID` (`InputID/outputID`) provides the unique ID to keep a record of the buffer given to the algorithm and released by the algorithm.

As explained above, buffer pointer cannot used as a unique identifier to keep a record of frame buffers. Any buffer given to algorithm should consider locked by algorithm, unless the buffer is returned to the application through `IVIDDEC3_OutArgs->freeBufID[]`.

> **Note:**
>
> `BufferID` returned in `IVIDDEC3_OutArgs ->outputID[]` is only for display purpose. Application should not consider it free unless it is a part of `IVIDDEC3_OutArgs->freeBufID[]`.

3.2.2 Frame Buffer Format

The frame buffer format to use for both progressive and interlaced pictures is as explained in the appendix on Picture Format.

3.2.3 Frame Buffer Management by Application

The application framework can efficiently manage frame buffers by keeping a pool of free frames from which it gives the decoder empty frames on request.

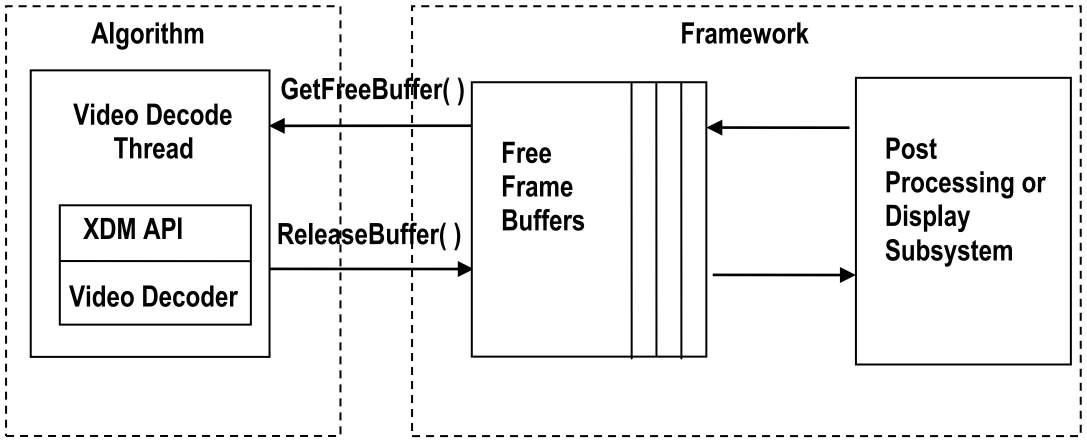


Figure 3-3. Interaction of Frame Buffers Between Application and Framework

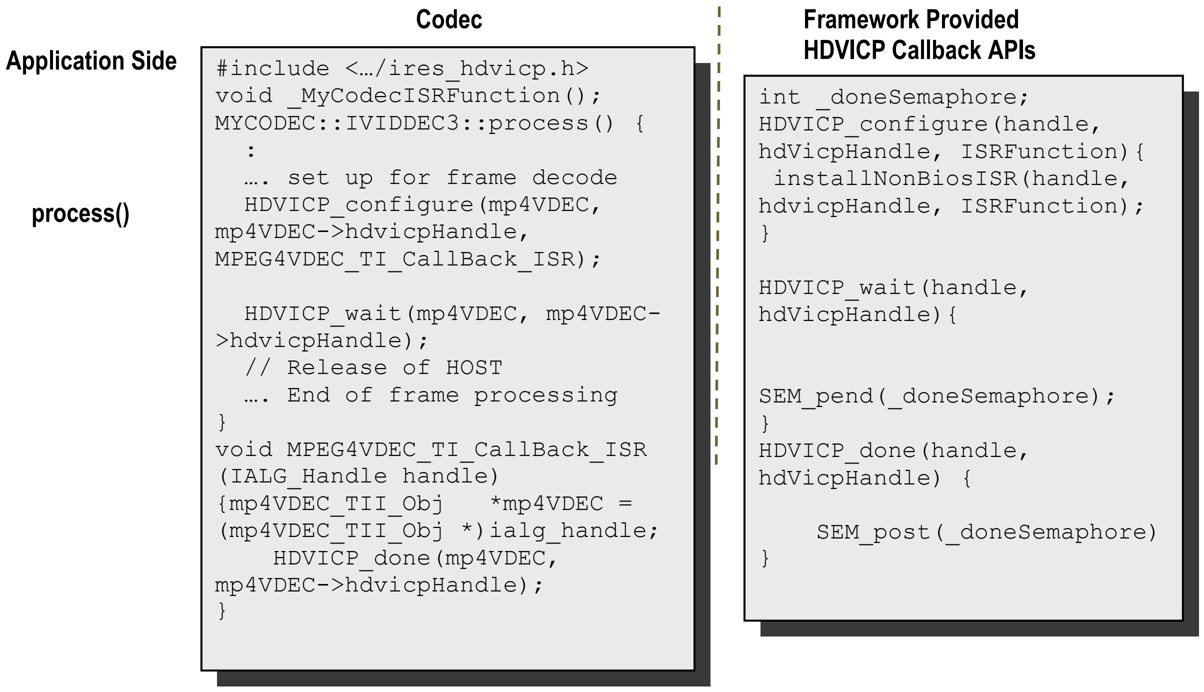
The sample application also provides a prototype for managing frame buffers. It implements the following functions. These functions are present in buffermanager.c provided along with test application.

- `BUFFMGR_Init()` - `BUFFMGR_Init` function is called by the test application to initialize the global buffer element array to default and to allocate the required number of memory data for reference and output buffers. The maximum required DPB size will set by the supported profile and level.

- `BUFFMGR_ReInit()` - `BUFFMGR_ReInit` function allocates global luma and chroma buffers and allocates entire space to the first element. This element will be used in the first frame decode. After the picture, height, width, and its luma and chroma buffer requirements will get. The global luma and chroma buffers will re-initialized to other elements in the buffer array.
- `BUFFMGR_GetFreeBuffer()` - `BUFFMGR_GetFreeBuffer` function searches for a free buffer in the global buffer array and returns the address of that element. If none of the elements is free, it will return NULL.
- `BUFFMGR_ReleaseBuffer()` - `BUFFMGR_ReleaseBuffer` function takes an array of buffer-IDs which are released by the test application. 0 is not a valid buffer ID, hence this function moves until it encounters a buffer ID as zero or it hits the `MAX_BUFF_ELEMENTS`.
- `BUFFMGR_DeInit()`- `BUFFMGR_DeInit` function releases all memory allocated by buffer manager.

3.3 Handshaking between Application and Algorithm

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



```
#include <…/ires_hdvicp.h>
void _MyCodecISRFunction();
MYCODEC::IVIDDEC3::process() {
  :
  …. set up for frame decode
  HDVICP_configure(mp4VDEC,
mp4VDEC->hdvicpHandle,
MPEG4VDEC_TI_CallBack_ISR);

  HDVICP_wait(mp4VDEC, mp4VDEC-
>hdvicpHandle);
  // Release of HOST
  …. End of frame processing
}
void MPEG4VDEC_TI_CallBack_ISR
(IALG_Handle handle)
{mp4VDEC_TII_Obj   *mp4VDEC =
(mp4VDEC_TII_Obj *)ialg_handle;
    HDVICP_done(mp4VDEC,
mp4VDEC->hdvicpHandle);
}
```

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

HDVICP_wait(handle,
hdVicpHandle){


SEM_pend(_doneSemaphore);
}
HDVICP_done(handle,
hdVicpHandle) {

    SEM_post(_doneSemaphore)
}
```

Figure 3-4. Interaction between Application and Codec.

Note:

- ❑ Process call architecture to share Host resource among multiple threads.
- ❑ ISR ownership is with the Host layer resource manager – outside the codec.
- ❑ The actual codec routine to be executed during ISR is provided by the codec.
- ❑ OS/System related calls (`SEM_pend, SEM_post`) also outside the codec.
- ❑ Codec implementation is OS independent.

The functions to implement by the application are:

- ❑ `HDVICP_configure(IALG_Handle handle, void *hdvicpHandle, void (*ISRfunctionptr)(IALG_Handle handle))`

  The algorithm to register its ISR function, which the application needs to call when it receives interrupts pertaining to the video task calls this function.

- ❑ `HDVICP_wait (void *hdvicpHandle)`

  The algorithm to move the video task to SEM-pend state calls this function.

- ❑ `HDVICP_done (void *hdvicpHandle)`

  The algorithm to release the video task from SEM-pend state calls this function. In the sample test application, these functions defined in hdvicp_framework.c file. The application can implement it in a way considering the underlying system.

3.4 Address Translations

The buffer addresses (DDR addresses) as seen by Media controller and HDVCIP2 (VDMA) will be different. Hence, address translations needed to convert from one address view to another. The application needs to implement a MEMUTILS function for this address translation (which will later implemented by the framework components). An example of the address translation function is as shown. The codec will make a call to this function from the host (Media Controller) library. Therefore, the function name and arguments should follow the example provided below. For a given input address, this function returns the VDMA view of the buffer (that is, address as seen by HDVCIP2).

```
Void *MEMUTILS_getPhysicalAddr(Ptr Addr)
{
 return ((void *)((unsigned int)Addr &
VDMAVIEW_EXTMEM));
}
```

Sample settings for the macro VDMAVIEW_EXTMEM is as shown

```
#if defined(HOST_ARM9)
  #define VDMAVIEW_EXTMEM      (0x07FFFFFF)
#elif defined(HOST_M3)
  #define VDMAVIEW_EXTMEM      (0xFFFFFFFF)
#else
  #define VDMAVIEW_EXTMEM      (0x07FFFFFF)
#endif
```

3.5 Sample Test Application

The test application exercises the IVIDDEC3 base class of the MPEG4 Decoder.

```
/*Main Function acting as a client for Video Decode Call*/

  BUFFMGR_Init();

  TestApp_SetInitParams(&params.viddecParams);

  /*---------------- Decoder creation -----------------*/
  handle = (IALG_Handle) mp4VDEC_create();
  /* Get Buffer information              */
  mp4VDEC_control(handle, XDM_GETBUFINFO);
  /* Do-While Loop for Decode Call  for a given stream  */
    do
    {
 /* Read the bitstream in the Application Input Buffer */
        validBytes = ReadByteStream(inFile);
       /* Get free buffer from buffer pool */
        buffEle = BUFFMGR_GetFreeBuffer();
/* Optional: Set Run-time parameters in the Algorithm via
control() */

       mp4VDEC_control(handle, XDM_SETPARAMS);

/*----------------------------------------------------*/
/* Start the process : To start decoding a frame      */
/*----------------------------------------------------*/
      retVal = mp4VDEC_decode(
                    handle,(XDM1_BufDesc *)&inputBufDesc,
                    (XDM_BufDesc *)&outputBufDesc,
                    (IVIDDEC3_InArgs *)&inArgs,
                    (IVIDDEC3_OutArgs *)&outArgs);

       /* Get the statatus of the decoder using comtrol */
       mp4VDEC_control(handle, XDM_GETSTATUS);

        /* Get Buffer information :      */
       mp4VDEC_control(handle, XDM_GETBUFINFO);


      /* Optional: Reinit the buffer manager in case the
      /* frame size is different           */
       BUFFMGR_ReInit();
      /* Always release buffers - which are released from
      /* the algorithm side -back to the buffer manager
*/
       BUFFMGR_ReleaseBuffer((XDAS_UInt32
*)outArgs.freeBufID);


}  while(1);
/* end of Do-While loop - which decodes frames         */

ALG_delete (handle);

BUFFMGR_DeInit();
```

Note:

This sample test application does not depict the actual function parameter or control code. It shows the basic flow of the code.

This page 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.

Topic **Page**

4.1 Symbolic Constants and Enumerated Data Types

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

Table 4-1. List of Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
`IVIDEO_FrameType`	`IVIDEO_NA_FRAME`	Frame type not available
	`IVIDEO_I_FRAME`	Intra coded frame
	`IVIDEO_P_FRAME`	Forward inter coded frame
	`IVIDEO_B_FRAME`	Bi-directional inter coded frame
	`IVIDEO_IDR_FRAME`	Intra coded frame that can be used for refreshing video content
	`IVIDEO_II_FRAME`	Interlaced Frame, both fields are I frames
	`IVIDEO_IP_FRAME`	Interlaced Frame, first field is an I frame, second field is a P frame
	`IVIDEO_IB_FRAME`	Interlaced Frame, first field is an I frame, second field is a B frame
	`IVIDEO_PI_FRAME`	Interlaced Frame, first field is a P frame, second field is a I frame
	`IVIDEO_PP_FRAME`	Interlaced Frame, both fields are P frames
	`IVIDEO_PB_FRAME`	Interlaced Frame, first field is a P frame, second field is a B frame
	`IVIDEO_BI_FRAME`	Interlaced Frame, first field is a B frame, second field is an I frame.
	`IVIDEO_BP_FRAME`	Interlaced Frame, first field is a B frame, second field is a P frame
	`IVIDEO_BB_FRAME`	Interlaced Frame, both fields are B frames
	`IVIDEO_MBAFF_I_FRAME`	Intra coded MBAFF frame
	`IVIDEO_MBAFF_P_FRAME`	Forward inter coded MBAFF frame
	`IVIDEO_MBAFF_B_FRAME`	Bi-directional inter coded MBAFF frame

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_MBAFF_IDR_FRAME	Intra coded MBAFF frame that used for refreshing video content.
	IVIDEO_FRAMETYPE_DEFAULT	Default set to IVIDEO_I_FRAME
IVIDEO_ContentType	IVIDEO_CONTENTTYPE_NA	Content type is not applicable
	IVIDEO_PROGRESSIVE IVIDEO_PROGRESSIVE_FRAME	Progressive video content
	IVIDEO_INTERLACED IVIDEO_INTERLACED_FRAME	Interlaced video content
	IVIDEO_INTERLACED_TOPFIELD	Interlaced video content, Top field
	IVIDEO_INTERLACED_TOPFIELD	Interlaced video content, Bottom field
	IVIDEO_CONTENTTYPE_DEFAULT	Default set to IVIDEO_PROGRESSIVE
IVIDEO_FrameSkip	IVIDEO_NO_SKIP	Do not skip the current frame. Default Value
	IVIDEO_SKIP_P	Skip forward inter coded frame. Not supported in current version of decoder.
	IVIDEO_SKIP_B	Skip bi-directional inter coded frame. Not supported in current version of decoder.
	IVIDEO_SKIP_I	Skip Intra coded frame. Not supported in current version of decoder.
	IVIDEO_SKIP_IP	Skip I and P frame/field(s) Not supported with current decoder version.
	IVIDEO_SKIP_IB	Skip I and B frame/field(s). Not supported with current decoder version.
	IVIDEO_SKIP_PB	Skip P and B frame/field(s). Not supported with current decoder version.
	IVIDEO_SKIP_IPB	Skip I/P/B/BI frames Not supported with current decoder version.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_SKIP_IDR	Skip IDR Frame Not supported with current decoder version.
	IVIDEO_SKIP_NONREFERENCE	Skip non reference frame Not supported with current decoder version..
	IVIDEO_SKIP_DEFAULT	Default set to IVIDEO_NO_SKIP
IVIDEO_VideoLayout	IVIDEO_FIELD_INTERLEAVED	Buffer layout is interleaved.
	IVIDEO_FIELD_SEPARATED	Buffer layout is field separated.
	IVIDEO_TOP_ONLY	Buffer contains only top field.
	IVIDEO_BOTTOM_ONLY	Buffer contains only bottom field
IVIDEO_OutputFrameStatus	IVIDEO_FRAME_NOERROR	Output buffer is available.
	IVIDEO_FRAME_NOTAVAILABLE	Codec does not have any output buffers.
	IVIDEO_FRAME_ERROR	Output buffer is available and corrupted.
	IVIDEO_OUTPUTFRAMESTATUS_DEFAULT	Default set to IVIDEO_FRAME_NOERROR
IVIDEO_PictureType	IVIDEO_NA_PICTURE	Frame type not available
	IVIDEO_I_PICTURE	Intra coded picture
	IVIDEO_P_PICTURE	Forward inter coded picture
	IVIDEO_B_PICTURE	Bi-directional inter coded picture
IVIDEO_DataMode	IVIDEO_FIXEDLENGTH	Input to the decoder is in multiples of a fixed length (example, 4K) (input side for decoder), Not supported with current decoder version.
	IVIDEO_SLICEMODE	Slice mode of operation (Input side for decoder). Not supported with current decoder version.
	IVIDEO_NUMROWS	Number of rows, each row is 16 lines of video (output side for decoder). Not supported with current decoder version.
	IVIDEO_ENTIREFRAME	Processing of entire frame data
IVIDEO_DataMode	IVIDEO_DECODE_ONLY	Decoding mode.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_ENCODE_ONLY	Encoding mode.
	IVIDEO_TRANSCODE_FRAMELEVEL	Transcode mode of operation encode/decode) which consumes/generates transcode information at the frame level. Not supported with current decoder version.
	IVIDEO_TRANSRATE_FRAMELEVEL	Transcode mode of operation encode/decode) which consumes/generates transcode information at the MB level. Not supported with current decoder version.
	IVIDEO_TRANSRATE_MBLEVEL	Transrate mode of operation encode/decode) which consumes/generates transcode information at the Frame level. Not supported with current decoder version.
	IVIDEO_TRANSCODE_MBLEVEL	Transrate mode of operation encode/decode) which consumes/generates transcode information at the MB level. Not supported with current decoder version.
IVIDDEC3_displayDelay	IVIDDEC3_DISPLAY_DELAY_AUTO	Decoder decides the display delay
	IVIDDEC3_DECODE_ORDER	Display frames are in decoded order without delay
	IVIDDEC3_DISPLAY_DELAY_1	Display the frames with 1 frame delay
	IVIDDEC3_DISPLAY_DELAY_2	Display the frames with 2 frame delay
	IVIDDEC3_DISPLAY_DELAY_3	Display the frames with 3 frame delay
	IVIDDEC3_DISPLAY_DELAY_4	Display the frames with 4 frame delay
	IVIDDEC3_DISPLAY_DELAY_5	Display the frames with 5 frame delay
	IVIDDEC3_DISPLAY_DELAY_6	Display the frames with 6 frame delay
	IVIDDEC3_DISPLAY_DELAY_7	Display the frames with 7 frame delay

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDDEC3_DISPLAY_DELAY_8	Display the frames with 8 frame delay
	IVIDDEC3_DISPLAY_DELAY_9	Display the frames with 9 frame delay
	IVIDDEC3_DISPLAY_DELAY_10	Display the frames with 10 frame delay
	IVIDDEC3_DISPLAY_DELAY_11	Display the frames with 11 frame delay
	IVIDDEC3_DISPLAY_DELAY_12	Display the frames with 12 frame delay
	IVIDDEC3_DISPLAY_DELAY_13	Display the frames with 13 frame delay
	IVIDDEC3_DISPLAY_DELAY_14	Display the frames with 14 frame delay
	IVIDDEC3_DISPLAY_DELAY_15	Display the frames with 15 frame delay
	IVIDDEC3_DISPLAY_DELAY_16	Display the frames with 16 frame delay
	IVIDDEC3_DISPLAYDELAY_DEFAULT	Same as IVIDDEC3_DISPLAY_DELAY_AUTO
XDM_DataFormat	XDM_BYTE	Big endian stream (default value)
	XDM_LE_16	16-bit little endian stream. Not supported with current decoder version.
	XDM_LE_32	32-bit little endian stream. Not supported with current decoder version.
	XDM_LE_64	64-bit little endian stream. Not supported with current decoder version..
	XDM_BE_16	16-bit big endian stream. Not supported with current decoder version.
	XDM_BE_32	32-bit big endian stream. Not supported with current decoder version.
	XDM_BE_64	64-bit big endian stream. Not supported with current decoder version.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
XDM_ChromaFormat	XDM_YUV_420P	YUV 4:2:0 planar. Not supported with current decoder version.
	XDM_YUV_422P	YUV 4:2:2 planar. Not supported with current decoder version.
	XDM_YUV_422IBE	YUV 4:2:2 interleaved (big endian). Not supported with current decoder version.
	XDM_YUV_422ILE	YUV 4:2:2 interleaved (little endian) (default value). Not supported with current decoder version.
	XDM_YUV_444P	YUV 4:4:4 planar Not supported with current decoder version..
	XDM_YUV_411P	YUV 4:1:1 planar. Not supported with current decoder version..
	XDM_GRAY	Gray format. Not supported with current decoder version.
	XDM_RGB	RGB color format. Not supported with current decoder version.
	XDM_YUV_420SP	YUV 4:2:0 chroma semi-planar. Supported, used as default and supported with current decoder version.
	XDM_ARGB8888	ARGB8888 color format. Not supported with current decoder version.
	XDM_RGB555	RGB555 color format. Not supported with current decoder version.
	XDM_RGB565	RGB565 color format. Not supported with current decoder version.
	XDM_YUV_444ILE	YUV 4:4:4 interleaved (little endian) color format. Not supported with current decoder version.
XDM_MemoryType	XDM_MEMTYPE_ROW	Raw Memory Type. Used as default.
	XDM_MEMTYPE_TILED8	2D memory in 8-bit container of tiled memory space ..
	XDM_MEMTYPE_TILED16	2D memory in 16-bit container of tiled memory space.
	XDM_MEMTYPE_TILED32	2D memory in 32-bit container of tiled memory space.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_MEMTYPE_TILEDPAGE	2D memory in page container of tiled memory space.
XDM_CmdId	XDM_GETSTATUS	Query algorithm instance to fill Status structure
	XDM_SETPARAMS	Set run-time dynamic parameters via the DynamicParams structure
	XDM_RESET	Reset the algorithm.
	XDM_SETDEFAULT	Initialize all fields in Params structure to default values specified in the library.
	XDM_FLUSH	Handle end of stream conditions. This command forces algorithm instance to output data without additional input.
	XDM_GETBUFINFO	Query algorithm instance regarding the properties of input and output buffers
	XDM_GETVERSION	Query the algorithm's version. The result will returned in the @c data field of the respective _Status structure.
	XDM_GETDYNPARAMSDEFAULT	Query algorithm instance regarding the dynamic parameters default values
XDM_AccessMode	XDM_ACCESSMODE_READ	The algorithm read from the buffer using the CPU. Used as default.
	XDM_ACCESSMODE_WRITE	The algorithm wrote from the buffer using the CPU
XDM_ErrorBit	XDM_PARAMSCHANGE	Bit 8 ❑ 1 - This error is applicable for transcoders. some key parameter of the input sequence changes ❑ 0 - Ignore
	XDM_APPLIEDCONCEALMENT	Bit 9 ❑ 1 - applied concealment ❑ 0 - Ignore
	XDM_INSUFFICIENTDATA	Bit 10 ❑ 1 - Insufficient data ❑ 0 - Ignore

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`XDM_CORRUPTEDDATA`	Bit 11 ❑ 1 - Data problem/corruption ❑ 0 - Ignore
	`XDM_CORRUPTEDHEADER`	Bit 12 ❑ 1 - Header problem/corruption ❑ 0 - Ignore
	`XDM_UNSUPPORTEDINPUT`	Bit 13 ❑ 1 - Unsupported feature/parameter in input ❑ 0 - Ignore
	`XDM_UNSUPPORTEDPARAM`	Bit 14 ❑ 1 - Unsupported input parameter or configuration ❑ 0 - Ignore
	`XDM_FATALERROR`	Bit 15 ❑ 1 - Fatal error ❑ 0 - Recoverable error
`IMPEG4VDEC_ErrorBit`	`IMPEG4D_ERR_VOS`	Bit 0 ❑ 1 - No Video Object Sequence detected in the frame ❑ 0 - Ignore
	`IMPEG4D_ERR_VO`	Bit 1 ❑ 1 - Incorrect Video Object type ❑ 0 - Ignore
	`IMPEG4D_ERR_VOL`	Bit 2 ❑ 1 - Error in Video Object Layer detected ❑ 0 - Ignore
	`IMPEG4D_ERR_GOV`	Bit 3 ❑ 1 - Error in Group of Video parsing ❑ 0 - Ignore
	`IMPEG4D_ERR_VOP`	Bit 4 ❑ 1 - Error in Video Object Plane parsing ❑ 0 - Ignore
	`IMPEG4D_ERR_SHORTHEADER`	Bit 5 ❑ 1 - Error in short header parsing ❑ 0 - Ignore
	`IMPEG4D_ERR_GOB`	Bit 6 ❑ 1 - Error in GOB parsing ❑ 0 - Ignore

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`IMPEG4D_ERR_VIDEOPACKET`	Bit 7 ❑ 1 - Error in Video Packet parsing ❑ 0 - Ignore
	`IMPEG4D_ERR_MBDATA`	Bit 16 ❑ 1 - Error in MB data parsing ❑ 0 - Ignore
	`IMPEG4D_ERR_INVALIDPARAM_IGNORE`	Bit 17 ❑ 1 - Invalid Parameter ❑ 0 - Ignore
	`IMPEG4D_ERR_UNSUPPFEATURE`	Bit 18 ❑ 1 - Unsupported feature ❑ 0 - Ignore
	`IMPEG4D_ERR_STREAM_END`	Bit 19 ❑ 1 - End of stream reached ❑ 0 - Ignore
	`IMPEG4D_ERR_VALID_HEADER_NOT_FOUND`	Bit 20 ❑ 1 – Valid header not found ❑ 0 – Ignore
	`IMPEG4D_ERR_UNSUPPRESOLUTION`	Bit 21 ❑ 1 - nsupported resolution by the decoder ❑ 0 - Ignore
	`IMPEG4D_ERR_BITSBUF_UNDERFLOW`	Bit 22 ❑ 1 - The stream buffer has underflowed ❑ 0 - Ignore
	`IMPEG4D_ERR_INVALID_MBOX_MESSAGE`	Bit 23 ❑ 1 - Invalid (unexpected) mail boX message recieved by HDVCIP2 ❑ 0 - Ignore
	`IMPEG4D_ERR_NO_FRAME_FOR_FLUSH`	Bit 24 ❑ 1 - Codec does not have any frame for flushing out to application ❑ 0 - ignore
	`IMPEG4D_ERR_VOP_NOT_CODED`	Bit 25 ❑ 1 - Given vop is not codec ❑ 0 - ignore

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IMPEG4D_ERR_START_CODE_NOT_PRESENT	Bit 26 ❑ 1 - Start code for given stream is not present in case of Parse Header mode ❑ 0 - ignore
	IMPEG4D_ERR_VOP_TIME_INCREMENT_RES_ZERO	Bit 27 ❑ 1 - Vop time increment resolution is zero ❑ 0 - ignore
	IMPEG4D_ERR_PICSIZECHANGE	Bit 28 ❑ 1 – resolution of the streams chanaged dynemically 0 - ignore
	IMPEG4D_ERR_UNSUPPORTED_H263_ANNEXS	Bit 29 ❑ 1 – decoder found unsupported annexs of h263 stream 0 - ignore
	IMPEG4D_ERR_HDVICP2_IMPROPER_STATE	Bit 30 ❑ 1 – status of HDVCIP2 is not in stand by state 0 - ignore
	IMPEG4D_ERR_FRAME_DROPPED	Bit 31 ❑ 1 – In sequence first frame is not I frame. 0 - ignore
IMPEG4VDEC_OptionalDeBlkMode	IMPEG4VDEC_DEBLOCK_DISABLE	This disables the out loop Deblocking filter
	IMPEG4VDEC_DEBLOCK_ENABLE	This enables the out loop Deblocking filter
	IMPEG4VDEC_ENHANCED_DEBLOCK_ENABLE	This enables enahanced out loop Deblocking filter
IMPEG4VDEC_DecodeOnlyIntraFrames	IMPEG4VDEC_DECODE_ONLY_I_FRAMES_DISABLE	This enables decoding of all frame types
	IMPEG4VDEC_DECODE_ONLY_I_FRAMES_ENABLE	This enables decoding of only Intra frames and skipping P/B frames

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:

- `XDM2_SingleBufDesc`
- `XDM2_BufDesc`
- `XDM1_AlgBufInfo`
- `IVIDEO2_BufDesc`
- `IVIDDEC3_Fxns`
- `IVIDDEC3_Params`
- `IVIDDEC3_DynamicParams`
- `IVIDDEC3_InArgs`
- `IVIDDEC3_Status`
- `IVIDDEC3_OutArgs`

4.2.1.1 XDM2_SingleBufDesc

|| Description

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

|| Fields

Field	Datatype	Input/ Output	Description
`*buf`	`XDAS_Int8`	Input	Pointer to the buffer
`memType`	`XDAS_Int32`	Input	Type of memory. See `XDM_MemoryType` enumeration for more details.
`usageMode`	`XDAS_Int16`	Output	Memory uses descriptor , set by application and used by algorithm
`bufSize`	`XDM2_BufSize`	Input	Size of the buffer(for tile memory/row memory)
`accessMask`	`XDAS_Int32`	input	If the buffer was not accessed by the algorithm processor (for example, it was

			filled by DMA or other hardware accelerator that does not write through the algorithm CPU), then bits in this mask should not be set.

4.2.1.2 XDM2_BufSize

‖ Description

This defines the union describing a buffer size.

‖ Fields

Field	Datatype	Input/ Output	Description
`width`	`XDAS_Int32`	Input	Width of buffer in 8-bit bytes. Required only for tile memory.
`height`	`XDAS_Int32`	Input	Height of buffer in 8-bit bytes. Required only for tile memory.
`Bytes`	`XDM2_BufSize`	Input	Size of the buffer in bytes, when tiled memory is not present then need to fill this by algorithm for buffer requirement in raw memory. If tiled memory is present then width and height should be filled instead of buffer requirement, By default Algorithm will fill width and height in Tiled memory, application will decide which kind of memory he is able to provide to codec.

4.2.1.3 XDM2_BufDesc

‖ Description

This structure defines the buffer descriptor for output buffers.

‖ Fields

Field	Datatype	Input/ Output	Description
`numBufs`	`XDAS_Int32`	Input	Number of buffers
`descs[XDM_MAX_IO_BUFFERS]`	`XDM2_SingleBufDesc`	Input	Array of buffer descriptors

4.2.1.4 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	Datatype	Input/ Output	Description
`minNumInBufs`	`XDAS_Int32`	Output	Number of input buffers
`minNumOutBufs`	`XDAS_Int32`	Output	Number of output buffers
`minInBufSize[XDM_MAX_IO_BUFFERS]`	`XDM2_BufSize`	Output	Size required for each input buffer
`minOutBufSize[XDM_MAX_IO_BUFFERS]`	`XDM2_BufSize`	Output	Size required for each output buffer
`inBufMemoryType[XDM_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	Memory type for each input buffer
`outBufMemoryType[XDM_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	Memory type for each output buffer
`minNumBufSets`	`XDAS_Int32`	Output	Minimum number of buffer sets for buffer management

Note:

For MPEG4 Advanced Simple Profile Decoder, the buffer details are:

- ❑ Number of input buffer required is 1.
- ❑ Number of output buffer required is 2 for YUV420 interleaved.
- ❑ If metadata is requested by the application, then see the Appendix B for buffer details.
- ❑ For frame mode of operation, there is no restriction on input buffer size except that it should contain atleast one frame of encoded data.
- ❑ The output buffer sizes (in bytes) for worst case 2048x2048 format are:
  - For YUV 420 interleaved:

    Y buffer = ((2048 + 32 + 127) & ~127) * (2048 + 32)
    UV buffer (((2048 + 32 + 127) & ~127) * (2048 + 32) >>1)

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

- ❑ The memory types supported for input buffers are `XDM_MEMTYPE_RAW` and `XDM_MEMTYPE_TILEDPAGE`.
- ❑ The memory types supported for luma output buffers are `XDM_MEMTYPE_TILED8`, `XDM_MEMTYPE_TILEDPAGE` and `XDM_MEMTYPE_RAW`
- ❑ The memory types supported for chroma output buffers are `XDM_MEMTYPE_TILED8`, `XDM_MEMTYPE_TILED16`,

XDM_MEMTYPE_TILEDPAGE and XDM_MEMTYPE_RAW.

4.2.1.5 IVIDEO2_BufDesc

|| Description

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

|| Fields

Field	Datatype	Input/ Output	Description
numPlanes	XDAS_Int32	Input/Output	Number of buffers for video planes
numMetaPlanes	XDAS_Int32	Input/Output	Number of buffers for Metadata
dataLayout	XDAS_Int32	Input/Output	Video buffer layout. See IVIDEO_VideoLayout enumeration for more details
planeDesc [IVIDEO_MAX_NUM_PLANES]	XDM1_SingleBufDesc	Input/Output	Description for video planes
metadataPlaneDesc [IVIDEO_MAX_NUM_METADATA_PLANES]	XDM1_SingleBufDesc	Input/Output	Description for metadata planes
secondFieldOffsetWidth[IVIDEO_MAX_NUM_PLANES]	XDAS_Int32	Input/Output	Off set value for second field in planeDesc buffer (width in pixels)
secondFieldOffsetHeight[IVIDEO_MAX_NUM_PLANES]	XDAS_Int32	Input/Output	Off set value for second field in planeDesc buffer (height in lines)
imagePitch	XDAS_Int32	Input/Output	Image pitch, common for all planes
imageRegion	XDM_Rect	Input/Output	Decoded image region including padding /encoder input image
activeFrameRegion	XDM_Rect	Input/Output	Actual display region/capture region
extendedError	XDAS_Int32	Input/Output	Provision for informing the error type if any
frameType	XDAS_Int32	Input/Output	Video frame types. See enumeration IVIDEO_FrameType. Not applicable for encoders
topFieldFirstFlag	XDAS_Int32	Input/Output	Indicates when the application (should display)/(had captured) the top field first. Not applicable for progressive content.
repeatFirstFieldFlag	XDAS_Int32	Input/O	Indicates when the first field should

Field	Datatype	Input/ Output	Description
		utput	be repeated. Not applicable for encoders.
`frameStatus`	`XDAS_Int32`	Input/O utput	Video in/out buffer status. Not applicable for encoders.
`repeatFrame`	`XDAS_Int32`	Input/O utput	Number of times to repeat the displayed frame. Not applicable for encoders.
`contentType`	`XDAS_Int32`	Input/O utput	Video content type. See `IVIDEO_ContentType`
`chromaFormat`	`XDAS_Int32`	Input/O utput	Chroma format for encoder input data/decoded output buffer. See `XDM_ChromaFormat` enumeration for details.
`scalingWidth`	`XDAS_Int32`	Input/O utput	Scaled image width for post processing for decoder. Not applicable for encoders.
`scalingHeight`	`XDAS_Int32`	Input/O utput	Scaled image height for post processing for decoder. Not applicable for encoders.
`rangeMappingLuma`	`XDAS_Int32`	Input/O utput	Applicable for VC1, set to -1 as default for other codecs
`rangeMappingChroma`	`XDAS_Int32`	Input/O utput	Applicable for VC1, set to -1 as default for other codecs
`enableRangeReductionFlag`	`XDAS_Int32`	Input/O utput	ON/OFF, default is OFF. Applicable only for VC1.

Note:

- `IVIDEO_MAX_NUM_PLANES`:
- Max YUV buffers - one for Y, and 1 for U and V interleaved.

The following parameters are not supported/updated in this version of the decoder

- `repeatFirstFieldFlag`
- `repeatFrame`
- `scalingWidth`
- `scalingHeight`
- `rangeMappingLuma`
- `rangeMappingChroma`
- `enableRangeReductionFlag`

4.2.1.6 IVIDDEC3_Fxns

|| Description

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

|| Fields

Field	Datatype	Input/ Output	Description
`Ialg`	`IALG_Fxns`	Input	Structure containing pointers for 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

4.2.1.7 IVIDDEC3_Params

|| Description

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

|| Fields

Field	Datatype	Input/ Output	Description
`Size`	`XDAS_Int32`	Input	Size of the basic or extended (if being used) data structure in bytes.
`maxHeight`	`XDAS_Int32`	Input	Maximum video height to be supported in pixels. Supported range is [64 to 2048] Default is 1088
`maxWidth`	`XDAS_Int32`	Input	Maximum video width to be supported in pixels. Supported range is [64 to 2048] Default is 1920
`maxFrameRate`	`XDAS_Int32`	Input	Maximum frame rate in fps * 1000 to be supported. Default is 30000
`maxBitRate`	`XDAS_Int32`	Input	Maximum bit-rate to be supported in bits per second. For example, if bit-rate is 10 Mbps, set this field to 10485760. Default is 10000000
`dataEndianness`	`XDAS_Int32`	Input	Endianness of input data. See `XDM_DataFormat` enumeration for details.

Field	Datatype	Input/ Output	Description
			Default is XDM_BYTE
forceChromaFormat	XDAS_Int32	Input	Sets the output to the specified format. Only 420 semi-planar format supported currently. For example, if the output should be in YUV 4:2:2 interleaved (little endian) format, set this field to XDM_YUV_422ILE. Default is XDM_YUV_420SP See XDM_ChromaFormat and eChromaFormat_t enumerations for details.
operatingMode	XDAS_Int32	Input	Video coding mode of operation (encode/decode/transcode/transrate). Only decode and transcode modes are supported in this version. Default value is IVIDEO_DECODE_ONLY Supported values are IVIDEO_DECODE_ONLY and IVIDEO_TRANSCODE_FRAMELEVEL
displayDelay	XDAS_Int32	Input	Display delay to start display. Default value is 1 (IVIDDEC3_DISPLAY_DELAY_1). Supported values are IVIDDEC3_DECODE_ORDER, IVIDDEC3_DISPLAY_DELAY_1 and IVIDDEC3_DISPLAY_DELAY_AUTO
inputDataMode	XDAS_Int32	Input	Input mode of operation. For decoder, it is fixed length/slice mode/entire frame. This version of the decoder supports only the entire frame mode - IVIDEO_ENTIREFRAME.
outputDataMode	XDAS_Int32	Input	Output mode of operation. For decoder, it is row mode/entire frame. This version of the decoder supports only the entire frame mode - IVIDEO_ENTIREFRAME
numInputDataUnits	XDAS_Int32	Input	Number of input slices/rows. For decoder, it is the number of slices or number of fixed length units. Default value is 0 Not supported in this version of the decoder. Value should be set to 0.
numOutputDataUnits	XDAS_Int32	Input	Number of output slices/rows. For decoder, it is the number of rows of output. Default value is 0

Field	Datatype	Input/ Output	Description
			Not supported in this version of the decoder. Value should be set to 0
`errorInfoMode`	`XDAS_Int32`	Input	Enable/disable packet error information for input/output. Supports only one value - `IVIDEO_ERRORINFO_OFF`
`displayBufsMode`	`XDAS_Int32`	Input	Indicates the `displayBufs` mode. This field can be set either as `IVIDDEC3_DISPLAYBUFS_EMBEDDED` or `IVIDDEC3_DISPLAYBUFS_PTRS`.
`metadataType`	`XDAS_Int32`	Input	Array of Metadata type. This field can be set either as `IVIDEO_METADATAPLANE_NONE` or `IVIDEO_METADATAPLANE_MBINFO` Default value is `IVIDEO_METADATAPLANE_NONE`

Note:

- MPEG4 Decoder does not use the `maxFrameRate` and `maxBitRate` fields for creating the algorithm instance. In the current implementation, `maxFrameRate` is set to 1000 * 30, and `maxBitRate` is set to 10000000.
- Maximum video height and width supported are 2048x2048
- `dataEndianness` field should be set to `XDM_BYTE`.
- The default value of `displayDelay` is 1.
- DataSync is not implemented so `inputDataMode` set as `IVIDEO_ENTIREFRAME`.
- `Data Sync` is not implemented so `outputDataMode` set as `IVIDEO_ENTIREFRAME`

4.2.1.8 IVIDDEC3_DynamicParams

|| Description

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

|| Fields

Field	Datatype	Input/ Output	Description

Field	**Datatype**	**Input/ Output**	**Description**
`Size`	`XDAS_Int32`	Input	Size of the basic or extended (if being used) data structure in bytes.
`decodeHeader`	`XDAS_Int32`	Input	Number of access units to decode, supported values are: ❑ 0 (`XDM_DECODE_AU`) - Decode entire frame including all the headers ❑ 1 (`XDM_PARSE_HEADER`) - Decode only one NAL unit (NA) Default value - 0 (`XDM_DECODE_AU`)
`displayWidth`	`XDAS_Int32`	Input	If the field is set to: 0 - Uses decoded image width as pitch If any other value greater than the decoded image width is given, then this value in pixels is used as pitch. Should be multiple of 128 bytes. Supported values – 0 & any value between 0 and maxwidth Default value is 0
`frameSkipMode`	`XDAS_Int32`	Input	Frame skip mode. See `IVIDEO_FrameSkip` enumeration for details. Default value is `IVIDEO_NO_SKIP`. No other value are supported.
`newFrameFlag`	`XDAS_Int32`	Input	Flag to indicate that the algorithm should start a new frame. Only value supported is `XDAS_TRUE`. This is useful for error recovery, for example, when the end of frame cannot be detected by the codec but is known to the application. Default value is `XDAS_TRUE`.
`*putDataFxn`	`XDM_DataSyncPutFxn`	Input	Function pointer to produce data at sub-frame level (`DataSync` call back function pointer for `putData`) Not supported in this version of the decoder. Default value is NULL.
`putDataHandle`	`XDM_DataSyncHandle`	Input	Handle that identifies the data sync FIFO and is passed as argument to `putData` calls Not supported in this version of the decoder. Default value is NULL.
`*getDataFxn`	`XDM_DataSyncGetFxn`	Input	Function pointer to receive data at sub-frame level (`DataSync` call back function pointer for `getData`) Not supported in this version of the decoder. Default value is NULL.

Field	Datatype	Input/ Output	Description
getDataHandle	XDM_DataSyncHandle	Input	Handle that identifies the data sync FIFO and is passed as argument to getData calls Not supported in this version of the decoder. Default value is NULL.
putBufferFxn	XDM_DataSyncPutBufferFxn	Input	Function pointer to receive buffer at sub-frame level Not supported in this version of the decoder. Default value is NULL.
putBufferHandle	XDM_DataSyncHandle	Input	Handle that identifies the data sync FIFO and is passed as argument to getBufferFxn calls. Not supported in this version of the decoder. Default value is NULL.
lateAcquireArg	XDAS_Int32	Input	Argument used during late acquire mode of the HDVCIP2 If the codec supports late acquisition of resources,and the application has supplied a lateAcquireArg value (via #XDM_SETLATEACQUIREARG), then the codec must also provide this lateAcquireArg value when requesting resources (i.e. during their call to acquire() when requesting the resource) Any value other than default value is ignored. Default value is IRES_HDVICP2_UNKNOWNLATEACQUIREARG

Note:

- Frame skip is not supported. Set the frameSkipMode field to IVIDEO_SKIP_DEFAULT.
- MPEG4 Decoder does not support newFrameFlag. It's value should be set as zero.

4.2.1.9 IVIDDEC3_InArgs

|| Description

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

|| Fields

Field	Datatype	Input/ Output	Description
Size	XDAS_Int32	Input	Size of the basic or extended (if being used) data

			structure in bytes.
numBytes	XDAS_Int32	Input	Size of input data (in bytes) provided to the algorithm for decoding
inputID	XDAS_Int32	Input	Application passes this ID to algorithm and decoder will attach this ID to the corresponding output frames. This is useful in case of re-ordering (for example, B frames). If there is no re-ordering, `outputID` field in the `VIDDEC3_OutArgs` data structure will be same as `inputID` field.

Note:

MPEG4 Decoder copies the `inputID` value to the `outputID` value of `IVIDDEC3_OutArgs` structure after factoring in the display delay.

4.2.1.10 IVIDDEC3_Status

|| Description

This structure defines parameters that describe the status of the decoder.

|| Fields

Field	**Datatype**	**Input/ Output**	**Description**
Size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
extendedError	XDAS_Int32	Output	Extended error code. See `XDM_ErrorBit` enumeration for details. The value of this parameter will be same as the value present in the `extendedError` parameter of the OutArgs
data	XDM1_SingleBufDesc	Output	Buffer information structure for information passing buffer.
maxNumDisplayBufs	XDAS_Int32	Output	Maximum number of buffers required by the codec.
maxOutArgsDisplayBufs	XDAS_Int32	Output	The maximum number of display buffers that can be returned through `IVIDDEC3_OutArgs.displayBufs`.
outputHeight	XDAS_Int32	Output	Output height in pixels
outputWidth	XDAS_Int32	Output	Output width in pixels
frameRate	XDAS_Int32	Output	Output frame rate

Field	Datatype	Input/ Output	Description
bitRate	XDAS_Int32	Output	Average bit-rate in bits per second
contentType	XDAS_Int32	Output	Video content. See IVIDEO_ContentType enumeration for details.
sampleAspectRatioHeight	XDAS_Int32	Output	Sample aspect ratio for height
sampleAspectRatioWidth	XDAS_Int32	Output	Sample aspect ratio for width
bitRange	XDAS_Int32	Output	Bit range. It is set to IVIDEO_YUVRANGE_FULL.
forceChromaFormat	XDAS_Int32	Output	Output chroma format. See XDM_ChromaFormat and eChromaFormat_t enumeration for details.
operatingMode	XDAS_Int32	Output	Mode of operation: Encoder/Decoder/Transcode/Transrate. It is set to IVIDEO_DECODE_ONLY.
frameOrder	XDAS_Int32	Output	Indicates the output frame order. This field is set to actual display delay value used by the decoder. Please see displayDelay in sec 4.2.1.7 for supported values.
inputDataMode	XDAS_Int32	Output	Input mode of operation. For decoder, it is fixed length/slice mode/entire frame. This version of the decoder supports only the fixed length and entire frame mode.
outputDataMode	XDAS_Int32	Output	Output mode of operation. For decoder, it is the row mode/entire frame. This version of the decoder supports only the entire frame mode.
bufInfo	XDM1_AlgBufInfo	Output	Input and output buffer information. See XDM1_AlgBufInfo data structure for details.
numInputDataUnits	XDAS_Int32	Input	Number of input data units i.e row/slice, ignored if entire frame has given as unit
numOutputDataUnits	XDAS_Int32	input	Number of input data units i.e row/slice, ignored if entire frame has given as unit
configurationID	XDAS_Int32	input	Configuration ID of given codec
metadataType	XDAS_Int32	input	Array of Metadata type plane

Field	Datatype	Input/ Output	Description
`decDynamicParams`	`IVIDDEC3_DynamicParams`	Output	Current values of the decoder's dynamic parameters.

> **Note:**
> - ❑ Algorithm sets the bit-Rate field to a default value 10485760.
> - ❑ The algorithm can set multiple bits of `extendedError` to 1, depending on the error condition

4.2.1.11 IVIDDEC3_OutArgs

‖ Description

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

‖ Fields

Field	Datatype	Input/ Output	Description
`size`	`XDAS_Int32`	Input	Size of the basic or extended (if being used) data structure in bytes.
`extendedError`	`XDAS_Int32`	Output	`extendedError` Field. The value of this parameter will be same as the value present in the `extendedError` parameter of the Status structure
`bytesConsumed`	`XDAS_Int32`	Output	Bytes consumed per decode call
`outputID[IVIDEO2_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	Output ID corresponding to `displayBufs` A value of zero (0) indicates an invalid ID. The first zero entry in array will indicate end of valid `outputID`s within the array. Hence, the application can stop reading the array when it encounters the first zero entry.
`decodedBufs`	`IVIDEO2_BufDesc`	Output	The decoder fills this structure with buffer pointers to the decoded frame. Related information fields for the Decoded frame are also populated. When frame decoding is not complete, as indicated by `outBufsInUseFlag`, the frame data in this structure will be incomplete. However, the algorithm will provide incomplete decoded frame data in case application may choose to use it for error recovery purposes.
`freeBufID[IVIDEO2_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	This is an array of `inputIDs` corresponding to the frames that have been unlocked in the

Field	Datatype	Input/ Output	Description
			current process call.
outBufsInUseFlag	XDAS_Int32	Output	Flag to indicate that the outBufs provided with the process () call are in use. No outBufs are required to be supplied with the next process () call.
displayBufsMode	XDAS_Int32	Output	Indicates the mode for #IVIDDEC3_OutArgs.displayBufs.
bufDesc [1]	IVIDEO2_Buf Desc	Output	Array containing display frames corresponding to valid ID entries in the outputID array. See IVIDEO2_BufDesc data structure for more details.
*pBufDesc[IVIDEO2_ MAX_IO_BUFFERS]	IVIDEO2_Buf Desc *	Output	Array containing pointers to display frames corresponding to valid ID entries in the @c outputID[]

Note:

- IVIDEO2_MAX_IO_BUFFERS - Maximum number of I/O buffers set to 20.The display buffer mode can be set as either IVIDDEC3_DISPLAYBUFS_EMBEDDED or IVIDDEC3_DISPLAYBUFS_PTRS.
- The current implementation of the decoder will always return a maximum of one display buffer per process call. If the mode is IVIDDEC3_DISPLAYBUFS_EMBEDDED, then the instance of the display buffer structure will be present in OutArgs. If the mode is IVIDDEC3_DISPLAYBUFS_PTRS, then a pointer to the instance will be present in OutArgs,
- The algorithm can set multiple bits of extendedError to 1, depending on the error condition [The extendedError of status structure will be in sync with the OutArgs structure's extendedError parameter].

4.2.2 MPEG4 Decoder Data Structures

This section describes the MPEG4 Decoder defined data structures, which are specific to MPEG4 Decoder. The MPEG4 Decoder structures can extend to define any specific parameters for supporting tools of MPEG4 Decoder. Below are the different data structure used by MPEG4 Decoder:-

- `IMPEG4VDEC_Params`
- `IMPEG4VDEC_DynamicParams`
- `IMPEG4VDEC_InArgs`
- `IMPEG4VDEC_Status`
- `IMPEG4VDEC_OutArgs`

4.2.2.1 IMPEG4VDEC _Params

|| Description

This structure defines the creation parameters and any other implementation specific parameters for an MPEG4 Decoder instance object. The creation parameters defined in the XDM data structure, `IVIDDEC3_Params`.

|| Fields

Field	Data Type	Input/ Output	Description
`viddec3Params`	`IVIDDEC3_Params`	Input	See `IVIDDEC3_Params` data structure for details.
`outloopDeBlocking`	`XDAS_Int32`	input	Flag to set by application for de-block filtering need be done by codec or not , default set to be zero. Supported Range: `IMPEG4VDEC_DEBLOCK_DISABLE=0`, `IMPEG4VDEC_DEBLOCK_ENABLE=1` & `IMPEG4VDEC_ENHANCED_DEBLOCK_ENABLE = 2`. Default Value: 0
`errorConcealmentEnable`	`XDAS_Int32`	input	Flag to set by application if concealment need to be done by codec in case of erroneous scenario Supported Range: 0 and 1. Default Value: 0
`sorensonSparkStream`	`XDAS_Int32`	Input	Reserved for future use Not used in this version of the decoder. Default value : 0

Field	Data Type	Input/ Output	Description
`debugTraceLevel`	`XDAS_UInt32`	Input	Specifies debug trace level. Supported Range: 0 to 2. Default Value: 0
`lastNFramesToLog`	`XDAS_UInt32`	Input	Specifies the number of most recent frames to log in debug trace. Supported Range: 0 to10. Default Value: 0

`paddingMode`	`XDAS_UInt32`	Input	Specify different methods of padding the for the reference frame when resolution of frame when dimension is non-multiple of 16.
			Supported values
			`PAD_METHOD_DIVX = 0,`
			`PAD_METHOD_MPEG4 = 1.`
			Default value is : `PAD_METHOD_DIVX`

`enhancedDeBlockingQp`	`XDAS_UInt32`	Input	Specifies the value of Qp used for enhanced out loop Deblocking filter. Will be valid if `outloopDeBlocking == IMPEG4VDEC_ENHANCED_DEBLOCK_ENABLE` only. The valid values are from 1 to 31 only.
`decodeOnlyIntraFrames`	`XDAS_UInt32`	Input	Flag to be set by application to request codec to decode only Intra frames and skip P/B frames. Supported Range: 0 & 1 Default Value: 0
`Reserved`	`XDAS_UInt32`	Input	Reserved for future use.

4.2.2.2 IMPEG4VDEC_DynamicParams

|| Description

This structure defines the run-time parameters and any other implementation specific parameters for an MPEG4 Decoder instance object. The run-time parameters defined in the XDM data structure, `IVIDDEC3_DynamicParams`.

|| Fields

Field	Data Type	Input/ Output	Description
`viddec3DynamicParams`	`IVIDDEC3_DynamicParams`	Input	See `IVIDDEC3_DynamicParams` data structure for details.

Field	Data Type	Input/ Output	Description
`Reserved[3]`	`XDAS_UInt32`	Input	Reserved for future use.

4.2.2.3 IMPEG4VDEC_InArgs

|| Description

This structure defines the run-time input arguments for an MPEG4 Decoder instance object.

|| Fields

Field	Data Type	Input/ Output	Description
`viddec3InArgs`	`IVIDDEC3_InArgs`	Input	See `IVIDDEC3_InArgs` data structure for details.

4.2.2.4 IMPEG4VDEC_Status

|| Description

This structure defines parameters that describe the status of the MPEG4 Decoder and any other implementation specific parameters. The `status` parameters defined in the XDM data structure, `IVIDDEC3_Status`.

|| Fields

Field	Data Type	Input/ Output	Description
`viddec3Status`	`IVIDDEC3_Status`	Output	See `IVIDDEC3_Status` data structure for details
`debugTraceLevel`	`XDAS_UInt32`	Output	Specifies the debug trace level. MPEG-4 Decoder supports till level 2.
`lastNFramesToLog`	`XDAS_UInt32`	Output	Specifies the number of most recent frames to log in debug trace.
`extMemoryDebugTraceAddr`	`XDAS_UInt32*`	Output	Address of the structure in external memory containing debug trace information
`extMemoryDebugTraceSize`	`XDAS_UInt32`	Output	Size of the structure containing the debug trace information

Field	Data Type	Input/ Output	Description
Reserved[3]	XDAS_UInt32	Output	Reserved for future use

4.2.2.5 IMPEG4VDEC_OutArgs

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
viddec3OutArgs	IVIDDEC3_OutArgs	Output	See IVIDDEC3_OutArgs data structure for details.
vopTimeIncrementResolution	XDAS_Int32	Output	vopTimeIncrementResolution indicates the number of evenly spaced subintervals, called ticks
vopTimeIncrement	XDAS_Int32	Output	vopTimeIncrement value represents the absolute vop_time_increment from the synchronization point marked by the modulo_time_base measured in the number of clock ticks.
mp4ClosedGov	XDAS_Int32	Output	mp4ClosedGov indicates the nature of the predictions used in the first consecutive B-VOPs (if any) immediately following the first coded I-VOP after the group of studio VOP header
mp4BrokenLink	XDAS_Int32	Output	mp4BrokenLink to indicate that the first consecutive B-VOPs (if any) immediately following the first coded I-frame following the group of studio VOP header may not be correctly decoded because the reference frame which is used for prediction is not available (because of the action of editing). A decoder may use this flag to avoid displaying frames that cannot be correctly decoded.

4.3 Interface Functions

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

- **Creation** – `algNumAlloc()`, `algAlloc()`
- **Initialization –** `algInit()`
- **Control** – `control()`
- **Data processing** – `algActivate()`, `process()`, `algDeactivate()`
- **Termination** – `algFree()`

You must call these APIs in the following sequence:

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

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

`algNumAlloc()`, `algAlloc()`, `algInit()`, `algActivate()`, `algDeactivate()`, and `algFree()` are standard XDAIS APIs. This document includes only a brief description for the standard XDAIS APIs. For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

4.3.1 Creation APIs

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

|| Name

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

|| Synopsis

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

|| Arguments

```
Void
```

|| Return Value

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

|| Description

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

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

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

|| See Also

`algAlloc()`

|| Name

`algAlloc()` – determine the attributes of all buffers that an algorithm requires

|| Synopsis

```
XDAS_Int32 algAlloc(const IALG_Params *params, IALG_Fxns
**parentFxns, IALG_MemRec memTab[]);
```

|| Arguments

```
IALG_Params *params; /* algorithm specific attributes */

IALG_Fxns **parentFxns;/* output parent algorithm
functions */

IALG_MemRec memTab[]; /* output array of memory records */
```

|| Return Value

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

|| Description

`algAlloc()` returns a table of memory records that describe the size, alignment, type, and memory space of all buffers required by an algorithm. If successful, this function returns a positive non-zero value indicating the number of records initialized.

The first argument to `algAlloc()` is a pointer to a structure that defines the creation parameters. This pointer may be `NULL;` however, in this case, `algAlloc()` must assume default creation parameters and must not fail.

The second argument to `algAlloc()` is an output parameter. `algAlloc()` may return a pointer to its parent's IALG functions. If an algorithm does not require a parent object to be created, this pointer must be set to `NULL`.

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

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

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

|| See Also

`algNumAlloc(), algFree()`

4.3.2 Initialization API

Initialization API is used to initialize an instance of the algorithm. The initialization parameters are defined in the `Params` structure (see Data Structures section for details).

|| Name

`algInit()` – initialize an algorithm instance

|| Synopsis

```
XDAS_Int32 algInit(IALG_Handle handle, IALG_MemRec
memTab[], IALG_Handle parent, IALG_Params *params);
```

|| Arguments

```
IALG_Handle handle; /* algorithm instance handle*/

IALG_memRec memTab[]; /* array of allocated buffers */

IALG_Handle parent; /* handle to the parent instance */

IALG_Params *params; /* algorithm initialization
parameters */
```

|| Return Value

```
IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */
```

|| Description

`algInit()` performs all initialization necessary to complete the run-time creation of an algorithm instance object. After a successful return `from algInit()`, the instance object is ready to be used to process data.

The first argument to `algInit()` is a handle to an algorithm instance. This value is initialized to the base field of `memTab[0]`.

The second argument is a table of memory records that describe the base address, size, alignment, type, and memory space of all buffers allocated for an algorithm instance. The number of initialized records is identical to the number returned by a prior call to `algAlloc()`.

The third argument is a handle to the parent instance object. If there is no parent object, this parameter must be set to `NULL`.

The last argument is a pointer to a structure that defines the algorithm initialization parameters.

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

|| See Also

`algAlloc(), algMoved()`

4.3.3 Control API

Control API is used for controlling the functioning of the algorithm instance during run-time. This is done by changing the status of the controllable parameters of the decoder during run-time. These controllable parameters are defined in the `Status` data structure (see Data Structures section for details).

|| Name

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

|| Synopsis

```
XDAS_Int32 (*control) (IVIDDEC3_Handle handle,
IVIDDEC3_Cmd id, IVIDDEC3_DynamicParams *params,
IVIDDEC3_Status *status);
```

|| Arguments

```
IVIDDEC3_Handle handle; /* algorithm instance handle */

IVIDDEC3_Cmd id; /* algorithm specific control commands*/

IVIDDEC3_DynamicParams *params /* algorithm run-time
parameters */

IVIDDEC3_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 `IVIDDEC3_DynamicParams` and `IVIDDEC3_Status` data structures respectively.

|| Preconditions

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

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

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

|| Postconditions

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

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

|| Example

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

|| See Also

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

4.3.4 Data Processing API

Data processing API is used for processing the input data.

|| Name

`algActivate()` – initialize scratch memory buffers prior to processing.

|| Synopsis

```
Void algActivate(IALG_Handle handle);
```

|| Arguments

```
IALG_Handle handle; /* algorithm instance handle */
```

|| Return Value

```
Void
```

|| Description

`algActivate()` initializes any of the instance's scratch buffers using the persistent memory that is part of the algorithm's instance object.

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

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

|| See Also

`algDeactivate()`

|| Name

`process()` – basic encoding/decoding call

|| Synopsis

```
XDAS_Int32 (*process)(IVIDDEC3_Handle handle, XDM1_BufDesc
*inBufs, XDM_BufDesc *outBufs, IVIDDEC3_InArgs *inargs,
IVIDDEC3_OutArgs *outargs);
```

|| Arguments

```
IVIDDEC3_Handle handle; /* algorithm instance handle */

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

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

IVIDDEC3_InArgs *inargs /* algorithm runtime input
arguments */

IVIDDEC3_OutArgs *outargs /* algorithm runtime output
arguments */
```

|| Return Value

```
IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */
```

|| Description

This function does the basic decoding/encoding. 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 `IVIDDEC3_InArgs` data structure that defines the runtime input arguments for an algorithm instance object.

The last argument is a pointer to the `IVIDDEC3_OutArgs` data structure that defines the runtime output arguments for an algorithm instance object.

|| Preconditions

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

- ❑ `process()` can only be called after a successful return from `algInit()` and `algActivate()`.
- ❑ If algorithm uses DMA resources, `process()` can only be called after a successful return from `DMAN3_init()`.
- ❑ `handle` must be a valid handle for the algorithm's instance object.
- ❑ Buffer descriptor for input and output buffers must be valid.
- ❑ Input buffers must have valid input data.

‖ Postconditions

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

- ❑ If the process operation is successful, the return value from this operation is equal to `IALG_EOK;` otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.
- ❑ After successful return from `process()` function, `algDeavtivate()` can be called.

‖ Example

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

‖ See Also

`algInit()`, `algDeActivate()`, `control()`

Note:

A video encoder or decoder cannot be pre-empted by any other video encoder or decoder instance. That is, you cannot perform task switching while encode/decode of a particular frame is in progress.

‖ 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* (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* (SPRU360).

‖ See Also

`algAlloc()`

Chapter 5

Frequently Asked Questions

This chapter provides answers to few frequently asked questions related to using this decoder.

5.1 Code Build and Execution

Question	Answer
How I will be able to run the codec library.	Please follow the instructions given in section **2.4 (Building and Running the Sample Test Application)**
Build error saying that code memory section is not sufficient	Make sure that project settings are not changed from the released package settings such as making project setting as File -03 and no debug information, which throws an error that code memory section is not sufficient.
Application returns an error saying “Couldn't open parameter file ….. “while running the host test app	Make sure that input file path is correct. If the application is accessing input from network, ensure that the network connectivity is stable.

5.2 Issues with Tools Version

Question	Answer
What tools are required to run the standalone codec?	To run the codec on standalone setup, you need Famework components, Code Composer Studio, ARM compiler tools (CG tools). If you are running on the simulator, then the correct version of the HDVCIP2 CSP is needed (See section 2.1 for more details.)

5.3 Algorithm Related

Question	Answer
What XDM interface does codec support?	Codec supports XDM IVIDDEC3 interface
Which Profile and level supported by this decoder?	This MPEG4 decoder support Advanced Simple Profile for levels 0,1,2,3,4,5 and 6 ; as well as it is also have support for simple profile for levels 0,1,2,3,4a, 5 and 6

Question	Answer
What kind of memory type supported for output buffer of the MPEG4 Decoder.	MPEG4 Decoder supports RAW(Luma-Chroma)/TILED8Bit(luma-chroma)/TILED16bit(chroma) and TILEDPAGE (Luam -chroma) output buffers, but MPEG4 Decoder does not support change in memory type dynamically in between process call. Means if codec has been provided RAW memory for process call 1, then codec expect Raw memory for the next process call, it should not be other than RAW mem.
Is MPEG4 Decoder support non-multiple of 16 frame height and width?	Yes-current decoder support for non-multiple of 16 frame height and width even for non-multiple 2 is also supported with this version.
Is this version of decoder having support for error concealment?	Yes current version of decoder having support for error concealment both spatial and temporal.
Is this version of decoder will support the display delay?	Only display delay of 1 or 0 frame supported in this release. Display delay 0 means decoding order.
What is the Maximum bit rate supported by this version of MPEG4 decoder?	This version of decoder supports up to 30Mbps.
Is this version of Decoder having support for Meta data parsing and provides the same to application?	Yes, current version of decoder is having Metadata support & provides to the application as well.
Is this version of decoder is having support for parse header functionality	Yes, with assumption that there will be byte-aligned boundary between header and residual data.
What are the resolutions supported by decoder?	Current version of decoder support picture resolution of minimum 64x64 and maximum 2048x2048 pixels , and also support for non-multiple of 2 resolution onward 64x64 picture size.
How input and output buffer provided by application are getting used by codec, does CPU directly operate on input and output buffer.	No, CPU does not operate directly on input and output buffer provided by application for reading the input data from input, it use VDMA to transfer the input data to its internal allocated buffer called SL2 memory and for writing/reading again it use VDMA.

Chapter 6

Debug Trace Usage

This section describes the debug trace feature supported by codec and its usage.

6.1 Introduction

This section explains the approach and overall design that will be adopted for enabling a trace from a video codec.

The primary use of Debug Trace Usage are:

1) Make the codec implementation capable of producing a trace containing details about the history of executing a particular instance of the codec

2) Enable the application to dump certain debug parameters from the codec in case of a failure. A failure might even be a hang or crash but in general can be defined as any unacceptable or erroneous behavior

Such a feature is targeted at providing more visibility into the operation of the codec and thus easing and potentially accelerating the process of debug.

6.2 Enabling and using debug information

To enable debug information, following two parameters are added to the create time parameters

1) debugTraceLevel

2) lastNFramesToLog

Hence the MPEG4 decoder create time parameters are modified as

```
typedef struct IMPEG4VDEC_Params

{

  IVIDDEC3_Params    viddec3Params;

  XDAS_Int32         outloopDeBlocking;

  XDAS_Int32         errorConcealmentEnable;

  XDAS_Int32         sorensonSparkStream;

  XDAS_UInt32        debugTraceLevel;

  XDAS_UInt32        lastNFramesToLog;
  XDAS_UInt32     paddingMode;

  XDAS_UInt32     enhancedDeBlockingQp

  XDAS_UInt32     reserved[2];

} IMPEG4VDEC_Params;
```

6.2.1 debugTracelevel

This parameter configures the codec to dump a debug trace log

- 0: Disables dumping of debug trace parameters
- >0: Enables the dumping of debug trace parameters. Value specifies the level of debug trace information. MPEG-4 decoder supports till level 2.

6.2.2 lastNFramesToLog

This parameter configures the codec to maintain history of debug trace parameters for last N frames.

- 0: No history will be maintained by the codec
- >0 : History of past specified number of frames will be maintained

In order to avoid book-keeping by the application to know whether the codec has been configured to dump debug trace and where the debug information is available, the following changes are done in the Status structure.

```
typedef struct IMPEG4VDEC_Status

{

  IVIDDEC3_Status  viddec3Status;

  XDAS_UInt32   debugTraceLevel;

  XDAS_UInt32   lastNFramesToLog;

  XDAS_UInt32 * extMemoryDebugTraceAddr;

  XDAS_UInt32   extMemoryDebugTraceSize;

  XDAS_UInt32   reserved [3];

} IMPEG4VDEC_Status;
```

debugTraceLevel: Debug trace level configured for the codec - 0, 1, 2

lastNFramesToLog: Number of frames for which history information is maintained by the codec

extMemoryDebugTraceAddr: External memory address (as seen by Media Controller) where debug trace information is being dumped – last memory buffer requested by the codec

extMemoryDebugTraceSize: External memory buffer size (in bytes) where debug trace information is being dumped - the size of last memory buffer

Now the application can retrieve this information from the codec at any time by the existing GETSTATUS query through the codec's Control API.

6.3 Debug Trace Levels

Debug trace has been (in this implementation) organized into 4 different levels arranged in a hierarchical fashion.

- Level 1 – Frame level information and profile data
- Level 2 – Slice and MB level information
- Level 3 – Logs function call stack for with entry hook
- Level 4 – Logs function call stack for with exit hook

At each higher level, the previous lower levels are also enabled. Please note MPEG-4 decoder supports up to debug trace level 2.

6.4 Requirements On The Application

The following are the requirements on the application side:

1. The application should be capable of configuring *debugTraceLevel* and *lastNFrameToLog* which are part of the Initialization Parameters of the codec
2. The application should be capable of querying the codec for its debug parameter memory regions and size
3. The application should be capable of retrieving these memory regions (In external memory or SL2) for the specified size and preserving these memory dumps in case of any erroneous behavior including a hang/crash.
4. The application, at any time (in case of hang, crash or any unexpected behavior) is expected to be also capable of retrieving the SL2 memory region as returned by the codec in Control-GETSTATUS specified by the SL2 memory debug trace address and size and provide it to the codec developer. The codec developer will have a PC based tool to parse and interpret this dump and produce a readable log of the debug trace parameters.

This page is intentionally left blank

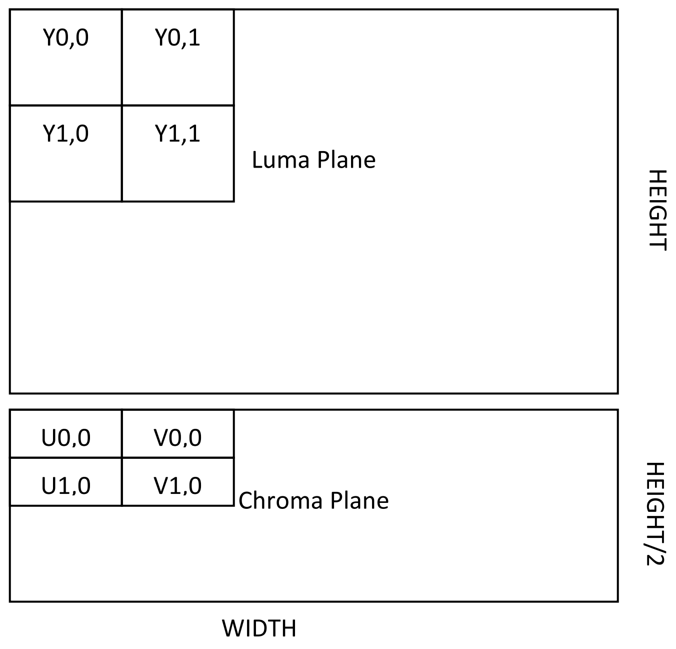
Appendix A

Picture Format

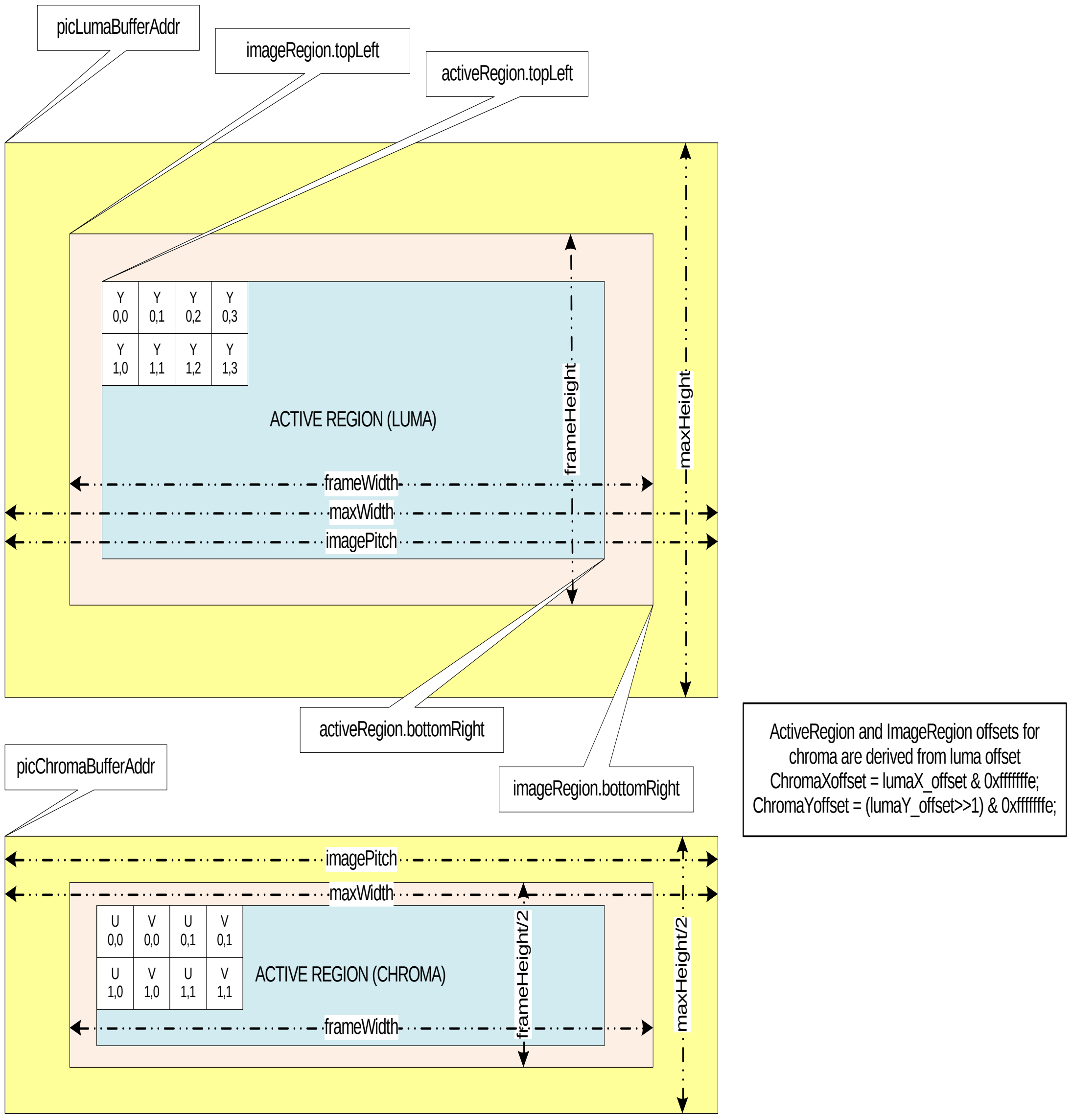
This Appendix explains picture format details for decoder. Decoder outputs YUV frames in NV 12 format.

A.1 NV12 Chroma Format

NV12 is YUV 420 semi-planar with two separate planes, one for Y, one for U and V interleaved.



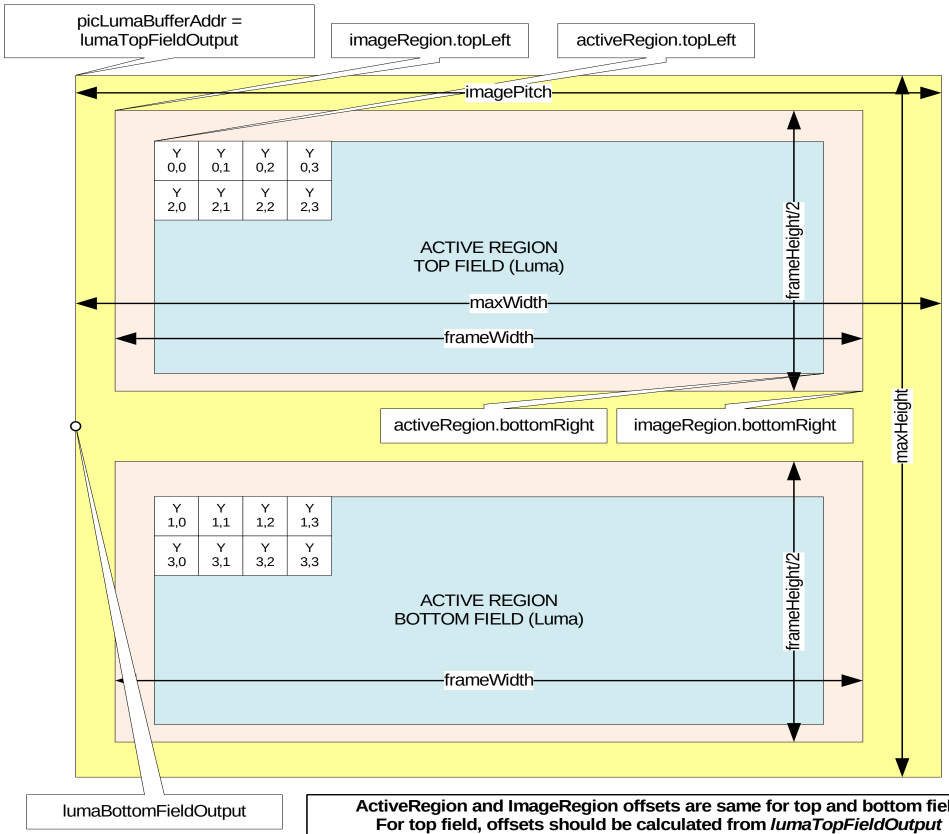
A.2 Progressive Picture Format



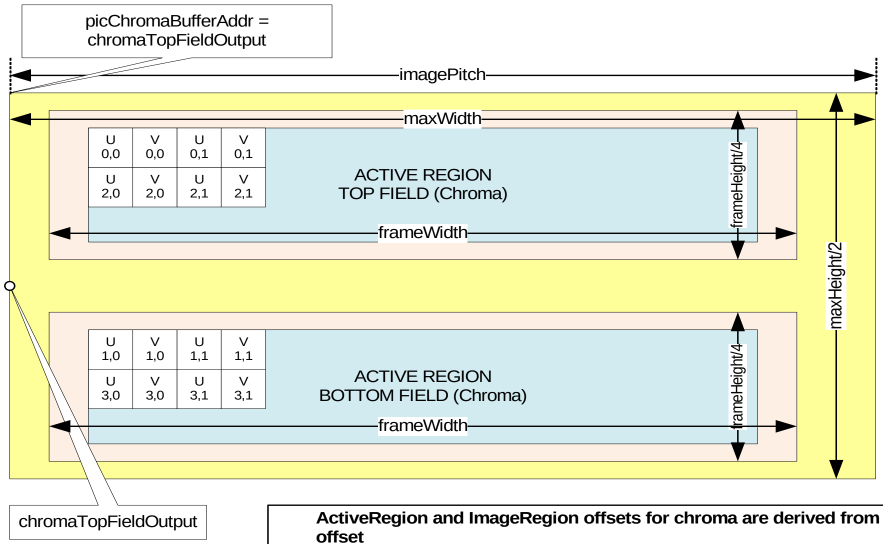
Note that for decoder in case of progressive sequence:

- Luma and chroma buffer addresses can be allocated independently
- Application shall provide this through separate buffer addresses
- The outermost yellow colored region is the minimum buffer that application should allocate for a given *maxWidth* and *maxHeight*
- *activeRegion*
    - The displayable region after cropping done by application.
- *imageRegion*
    - Image data decoded by the decoder whose dimensions are always multiple of 16.
    - Contains the active Region as a proper subset.
- *Picture Buffer (pic(Luma/Chroma)BufferAddr)*
    - Contains padded regions and extra region due to alignment constraints.
    - Contains the imageRegion as a proper subset.
- *imagePitch*
    - The difference in addresses of two vertically adjacent pixels
    - Typically equal to width of the picture Buffer.
- *Padding Amounts*
    - In vertical direction (top and bottom), padding amount is 16 pixels for Luma buffer and 8 pixels for chroma buffer.
    - In horizontal direction (left and right), padding amount is 16 pixels for both Luma buffer chroma buffer.

A.3 Interlaced Picture Format



ActiveRegion and ImageRegion offsets are same for top and bottom field
For top field, offsets should be calculated from *lumaTopFieldOutput*
For bottom field, offsets should be calculated from *lumaBottomFieldOutput*



ActiveRegion and ImageRegion offsets for chroma are derived from luma offset
ChromaXoffset = lumaX_offset & 0xfffffffe;
ChromaYoffset = (lumaY_offset >> 1) & 0xfffffffe;
ActiveRegion and ImageRegion offsets are same for top and bottom field
For top field, offsets should be calculated from *lumaTopFieldOutput*
For bottom field, offsets should be calculated from *lumaBottomFieldOutput*

Note that for decoder in case of interlaced sequence:

- Luma and chroma buffers can be allocated independently
- Field buffer allocation cannot be independent
- For every pair of top and bottom field, decoder shall expect a single buffer address from the application
- Decoder will not give separate decoded field as output, instead of this decoder will give complete interleaved fields decoded output in one process call.
- The outermost yellow coloured region is the minimum buffer that application should allocate for a given *maxWidth* and *maxHeight*
- *activeRegion*
    - The displayable region after cropping done by application.
- *imageRegion*
    - Image data decoded by the decoder.
    - Contains the activeRegion as a proper subset.
- *Picture Buffer (pic(Luma/Chroma)Buffered)*
    - Contains padded regions and extra region due to alignment constraints.
    - Contains the image Region as a proper subset.
- *imagePitch*
    - The difference in addresses of two vertically adjacent pixels
    - Typically equal to width of the picture Buffer.
- *Padding Amounts*
    - In vertical direction (top and bottom), for each field, padding amount is 16 pixels for Luma buffer and 8 pixels for chroma buffer.
    - In horizontal direction (left and right), padding amount is 16 pixels for both Luma buffer chroma buffer.

A.4 Constraints on Buffer Allocation for Decoder

- *maxWidth* and *maxHeight* are inputs given by the decoder to the applications
  - Application may not know the output format of the decoder.
  - Therefore, application should allocate Image Buffer based on *maxWidth* and *maxHeight*
    - The extra region beyond the (maxWidth x maxHeight) requirements may be allocated by application due to alignment, pitch or some other constraints
- Application needs to ensure following conditions regarding *imagePitch*
  - *imagePitch* shall be greater or equal to the maxWidth.
  - *imagePitch* shall be multiple of 128 bytes (if the buffer is not in TILED region).
  - *imagePitch* shall actually be the tiler space width (i.e. depends on how many bit per pixel, for 8bpp 16bpp and 32bpp respectively 16Kbyte, 32Kbyte and 32Kbyte). (if the buffer is in TILED region).
  - Application may set *imagePitch* greater than *maxWidth* as per display constraints. However this value must be a multiple of 128 bytes (if the buffer is not in TILED region).
- *picLumaBufferAddr* and *picChromaBufferAddr* shall be 16-byte aligned address. (if the buffer is not in TILED region).
- *ActiveRegion.topLeft* and *ActiveRegion.bottomRight* are decoder outputs
  - Application should calculate actual display width and display height based on these parameters
  - *ActiveRegion.topLeft* and *ActiveRegion.bottomRight* shall be identical for both fields in case of interlaced format
- Maximum and Minimum Resolution is defined as below
  - Progressive
    - Minimum frameWidth = 64
    - Minimum frameHeight = 64
    - Maximum frameWidth = 2048
    - Maximum frameHeight = 2048
  - Interlaced
    - Minimum frameWidth = 64
    - Minimum (frameHeight / 2) = 32
    - Maximum frameWidth = 2048
    - Maximum (frameHeight / 2) = 1024
- Typically picture buffer allocation requirements for decoder, after buffer addresses meet alignment constraints (depends on decoder's padding requirements), for both progressive and interlaced are as given below.
  - Luma buffer size = maxWidth x maxHeight and

    Chroma buffer size = maxWidth x maxHeight/2 where
    - maxWidth = frameWidth + 32 (progressive/interlaced)

- maxHeight = frameHeight + 32 (progressive)
- maxHeight = frameHeight + 32 (interlaced)

This page intentionally left blank

Appendix B

Meta Data Support

This version of the decoder supports writing out the MB Info data into application provided buffers.

This feature can be enabled/disabled through create time parameters `IVIDDEC3_Params::metadataType [IVIDEO_MAX_NUM_METADATA_PLANES]`. There can be maximum 3 (`IVIDEO_MAX_NUM_METADATA_PLANES`) meta data planes possible to be supported with one instance of the decoder.

Each element of `metadataType[]` array can take following enumerated values.

Enumeration	Value
IVIDEO_METADATAPLANE_NONE	-1
IVIDEO_METADATAPLANE_MBINFO	0
IVIDEO_METADATAPLANE_EINFO	1
IVIDEO_METADATAPLANE_ALPHA	2

This version of the decoder supports only following enumerated values:

1) `IVIDEO_METADATAPLANE_NONE`
2) `IVIDEO_METADATAPLANE_MBINFO`

If user does not want to use any meta data plane then all the entries of `IVIDDEC3_Params::metadataType[]` should be set to `IVIDEO_METADATAPLANE_NONE`. Note that the `metadataType[]` array need to be filled contiguously (there cannot be `IVIDEO_METADATAPLANE_NONE` between two metadata types.

The buffer requirements for metadata can be obtained using Control call with XDM_GETBUFINFO:

The buffer pointers for the metadata need to be supplied as below during process Call:

- ❑ `OutBufs->numBufs = numBuffers` forYUVPlanes + number of meta data enabled (This is =3 if Mb-info metadata is enabled)

  - `outBufs->descs[0]` -> Y plane
  - `outBufs->descs[1]` -> Cb/Cr plane outBufs.
  - `outBufs->descs[2]` -> Buffer allocated for MB info

- Also, the respective buffer pointer is copied back in the first meta-plane pointer: `outArgs->decodedBufs.metadataPlaneDesc[0].buf` , again the ordering of the metadata is as per the order supplied by `IVIDDEC3_Params::metadataType[] inpput parameter.`

Decoder parses metadata in the current process call and returns in the same process call. This means, effectively Meta data will be given out in decode order [Not in Display Order]. If application is interested in display order, it should have logic to track based on input and output ID. In case of interlaced pictures, Meta data buffers provided for each field (each process call) is assumed to be independent.

3) **Remainder of this Appendix gives more information about** `IVIDEO_METADATAPLANE_MBINFO`

Decoder shares two types of information at MB Level:

MB Error Map: It is an array of bytes - One byte per MB (Refer Enum IH264VDEC_ mbErrStatus). The byte indicates whether the MB is in error or not.

MB Info structure: It is a structure, which defines properties of a MB. Refer structure IMPEG4VDEC_TI_MbInfo in impeg4vdec.h file. Size per MB = 112 bytes.

Case1: If the Application sets viddec3Params.metadataType[x] = IVIDEO_METADATAPLANE_MBINFO and IVIDDEC3_Params.operatingMode = IVIDEO_DECODE_ONLY, then decoder will dump out MB Error Map and error concealment structure at buffer location given for MB Info meta data.

Case2: If the Application sets viddec3Params.metadataType[x] = IVIDEO_METADATAPLANE_MBINFO and IVIDDEC3_Params.operatingMode = IVIDEO_TRANSCODE_FRAMELEVEL, then decoder will dump out MB Error Map at buffer location given for MB Info Meta data. Error Map will be followed by MB Info structure for all MBs.

Note that if the Application does not set viddec3Params.metadataType[x] = IVIDEO_METADATAPLANE_MBINFO, then no information will be dumped, irrespective of the value of IVIDDEC3_Params.operatingMode. In addition, as a minor Interface limitation, there is no provision to dump MB Info structure alone w/o error map.

Format details for Case 1 (Dumping of Error map):

Case 1a, Progressive Frame:

Error Map, Size in Bytes = Number of MBs in Frame

Case 1b, Interlaced Frame:

Error Map, Size in Bytes = Number of MBs in Frame

Format details for Case 2 (Dumping of Error map and MB Info):

Case 2a, Progressive Frame:

Error Map, Size in Bytes = Number of MBs in Frame

MB Info structure for all MBs, Size in Bytes = 112 * Number of MBs in Frame

Case 2b, Interlaced Frame:

Error Map, Size in Bytes = Number of MBs in Frame

MB Info structure for all MBs, Size in Bytes = 112 * Number of MBs in Frame

How to configure codec to give MBInfo data?

Set the `operatingMode` of `IVIDDEC3_Params` to `IVIDEO_TRANSCODE_FRAMELEVEL`.

Then populate the `oubufs` as mentioned above.

This page intentionally left blank

Appendix C

Error Handling

This version of the decoder supports handling of erroneous situations while decoding. If decoder encounters errors in bit stream or any other erroneous situations, decoder shall exit grace fully without any hang or crash. In addition, decoder process call shall return `IVIDDEC3_EFAIL` and relevant error code will be populated in `extendedError` field of `outArgs`. Different error codes and their meanings are described below.

Definitions of bits numbered 8-15 are as per common XDM definition. Definition of remaining bits are MPEG4 Decoder specific and as given in below tabular column. Bit numbering in the 32 bit word `extendedError` is from Least Significant Bit to Most Significant Bit.

Some of the erroneous situations will be reported as `XDM_FATALERROR` by the decoder. In these cases, Application should perform XDM_RESET of the decoder or re-create the decoder. After an `XDM_RESET` is performed or re-created, the decoder will treat the bit stream provided freshly and it shall use no information from previously parsed data.

In certain fatal erroneous situations, the Application, might flush out the locked buffers, if need be. See below table for more details on error situations when flush can be performed.

In case of non-fatal errors, application need not perform `XDM_RESET`. It can proceed with more decode calls, if bit stream is still not exhausted.

Meanings of various error codes and the recommended application behavior are provided in the following table:

Table 6-2. Error Codes Information

Bit	Error code	Explanation	XDM Error Code Mapping	Recommended App Behavior
0	`IMPEG4D_ERR_VOS`	Any syntax error while parsing the visual object sequence of mpeg4 stream.	`XDM_CORRUPTEDHEADER`	If more bytes available in bit stream, then pass it to decoder. ELSE, if bytes are not available call Flush operation.
1	`IMPEG4D_ERR_VO`	Any syntax error while parsing the visual object of the mpeg4 stream.	`XDM_CORRUPTEDHEADER`	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.

Bit	Error code	Explanation	XDM Error Code Mapping	Recommended App Behavior
2	IMPEG4D_ERR_VOL	Any syntax error while parsing the video object layer of mpeg4 stream.	XDM_CORRUPTEDHEADER	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.
3	IMPEG4D_ERR_GOV	Any syntax error while parsing the group of vop of mpeg4 stream.	XDM_CORRUPTEDHEADER	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.
4	IMPEG4D_ERR_VOP	Any syntax error while parsing the video object packet of the mpeg4 stream.	XDM_CORRUPTEDHEADER	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.
5	IMPEG4D_ERR_SHORTHEADER	Any syntax error while parsing the H.263 stream frame header.	XDM_CORRUPTEDHEADER	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.
6	IMPEG4D_ERR_GOB	Any syntax error while parsing GOB in case of H.263 stream.	XDM_CORRUPTEDHEADER	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.
7	IMPEG4D_ERR_VIDEOPACKET	Any syntax error while parsing video packet of the stream.	XDM_CORRUPTEDHEADER	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.
8	XDM_PARAMSCHANGE	Video object layer gets changed	XDM_PARAMSCHANGE	Refer codec specific error which causes this
9	XDM_APPLIEDCONCEALMENT	Applied concealment	XDM_APPLIEDCONCEALMENT	Refer codec specific error which causes this

Bit	Error code	Explanation	XDM Error Code Mapping	Recommended App Behavior
10	XDM_INSUFFICIENTDATA	Insufficient input data	XDM_INSUFFICIENTDATA	Refer codec specific error which causes this
11	XDM_CORRUPTEDDATA	Data problem/corruption	XDM_CORRUPTEDDATA	Refer codec specific error which causes this
12	XDM_CORRUPTEDHEADER	Header problem/corruption	XDM_CORRUPTEDHEADER	Refer codec specific error which causes this
13	XDM_UNSUPPORTEDINPUT	Unsupported feature/parameter	XDM_UNSUPPORTEDINPUT	Refer codec specific error which causes this
14	XDM_UNSUPPORTEDPARAM	Unsupported input parameter	XDM_UNSUPPORTEDPARAM	Refer codec specific error which causes this
15	XDM_FATALERROR	Fatal error	XDM_FATALERROR	Refer codec specific error, which causes this, but in this case application need to do the XDM_RESET or may re-create the decoder and give fresh stream for decoding.
16	IMPEG4D_ERR_MBDATA	Any syntax error while parsing the Mb header or coefficient data.	XDM_CORRUPTEDDATA	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.
17	IMPEG4D_ERR_INVALIDPARAM_IGNORE	Some error was detected while slice header decoding, which the codec corrected and Continued. Application should Ignore this error.	XDM_CORRUPTEDHEADER	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.
18	IMPEG4D_ERR_UNSUPPFEATURE	Some unsupported feature set reported in stream while parsing. Example GMC/sprite coding	XDM_FATALERROR / XDM_UNSUPPORTEDINPUT	Can either continue with the stream giving a fresh pointer OR do XDM Reset and give a fresh stream, depending on other XDM

Bit	Error code	Explanation	XDM Error Code Mapping	Recommended App Behavior
		etc…		error bit set.
19	IMPEG4D_ERR_STREAM_END	End of Stream was found in this process call OR codec is in flush mode	No XDM mapping	Normal Mode of Decoder - Do XDM_FLUSH, Else - XDM_RESET and Next Stream
20	IMPEG4D_ERR_VALID_HEADER_NOT_FOUND	In current process call decoder could not get any valid header of mpeg4 or H.263 stream	XDM_CORRUPTEDHEADER	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.
21	IMPEG4D_ERR_UNSUPPRESOLUTION	Width or height is less than the minimum supported or more than the maximum supported	XDM_FATALERROR / XDM_UNSUPPORTEDINPUT	Can do a FLUSH, then XDM Reset and pass a fresh stream
22	IMPEG4D_ERR_BITSBUF_UNDERFLOW	In current process call given are not sufficient for decoding the current frame.	XDM_INSUFFICIENTDATA	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.
23	IMPEG4D_ERR_INVALID_MBOX_MESSAGE	Invalid message received on MB, which causes interrupt on Media Controller or HDVCIP2, depending on the FIFO - Stray writes into FIFO by some one other than codec	XDM_FATALERROR	Should not do XDM_FLUSH. Do HDVICP_Reset, XDM Reset and pass stream
24	IMPEG4D_ERR_NO_FRAME_FOR_FLUSH	Decoder does not have any valid decoded frame data for flush.	XDM_UNSUPPORTEDINPUT	Do the XDM Reset and pass a fresh stream, or pass the same stream but with valid input and output buffer descriptors.
25	IMPEG4D_ERR_VOP_NOT_CODED	Current process call encounter the scenario that current frame is not coded.	XDM_CORRUPTEDHEADER	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.

Bit	Error code	Explanation	XDM Error Code Mapping	Recommended App Behavior
26	IMPEG4D_ERR_START_CODE_NOT_PRESENT	No valid start code present in the stream, this error code is mainly set for parse header mode	XDM_CORRUPTEDHEADER	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush operation.
27	IMPEG4D_ERR_VOP_TIME_INCREMENT_RES_ZERO	Decoder found vop time increment resolution is zero while parsing the VOL	XDM_UNSUPPORTEDPARAM / XDM_FATALERROR	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush.
28	IMPEG4D_ERR_PICSIZECHANGE	Decoder found that resolution of the streams gets changed dynamically while parsing the VOL	XDM_UNSUPPORTEDPARAM / XDM_FATALERROR	If more bytes available in bit stream, then pass it to decoder. ELSE if bytes are not available call Flush.
29	IMPEG4D_ERR_UNSUPPORTED_H263_ANNEXS	Decoder found unsupported annexes while parsing the VOP header of the h263 stream.	XDM_UNSUPPORTEDPARAM / XDM_FATALERROR	It is unsupported feature by current decoder, so reset/re-create the decoder and run other media file.
30	IMPEG4D_ERR_HDVICP2_IMPROPER_STATE	HDVICP2 is not in proper state	XDM_FATALERROR	It is unexpected state of the HDVCIP2, so make sure that HDVCIP2 is in stand by mode before giving control to Decoder. Codec needs to be re-created.
31	IMPEG4D_ERR_IFRAME_DROPPED	In sequence first frame (I Frame) is not present	No XDM mapping	In case codec has found this error, functionally codec decoding will not gets affected, only decoded output may not be correct.

This page is intentionally left blank

Appendix D

Parse Header Support

This version of the decoder provides support to parse just header of the bit stream. For decoder to operate in this mode Application needs to perform a `XDM_SETPARAMS` control call with `dynamicParams-> decodeHeader = XDM_PARSE_HEADER`

Typical usage of this feature by the application is to understand the resolution of picture in bit stream and allocate frame buffer of size as needed by that bit stream. Sequence of operations on the application side typically is as follows:

1. Decoder_Create
2. Control call (XDM_SETPARAMS) to configure decoder in parse header mode
3. Process call to decoder which shall decode VOS + VO + VOL + VOP Header
4. Control call (XDM_GETBUFINFO) to understand buffer requirements
5. Allocate buffers of size exactly needed to decode this particular bit stream
6. Control call (XDM_SETPARAMS) to configure decoder in normal mode (dynamicParams->decodeHeader = XDM_DECODE_AU)
7. Process calls to decode frames

Note:

Following aspects of decoder behavior when configured in `XDM_PARSE_HEADER` mode:

- Decoder shall decode VOS/VO/VOL of the mpeg4 stream, if VOL is not present in stream then first it will try to get the VOL or valid h.263 header and if codec gets VOL ,it will treat up coming stream as mpeg else if it gets h.263 header first then codec treat stream as h263 stream.
- After at least one VO/VOS/VOL is parsed, if application still performs process calls with decoder in `XDM_PARSE_HEADER` mode, then decoder behavior is as follows:
- If decoder encounters VO/VOS/VOL, it shall parse them until vop data is encountered. After encountering vop data, it shall return from process call
- Output buffers for YUV data is don't care for decoder, while in parse header mode

This page is intentionally left blank

Appendix E

Support for Display Delay

This version of decoder supports configurability to achieve desired display delay and low DDR memory footprint.

It is recommended to utilize this feature only when the application is well aware of the nature of the bit stream in terms of the GOP structure.

Desired display delay can be achieved by the application by setting IVIDDEC3_Params::displayDelay. Decoder shall start displaying of frames not later than display Delay numbers of frames are decoded.

Note:

- MPEG4 Decoder supports only three mode of displayDelay parameter of IVIDDEC3_Params
- `IVIDDEC3_DISPLAY_DELAY_AUTO`, decoder internally set display delay as 1 frame delay, so that decoded data in current process call will get displayed in next process call.
- `IVIDDEC3_DISPLAY_DELAY_1`, decoder get set by display delay as 1 frame delay, so that decoded data in current process call will get displayed in next process call
- `IVIDDEC3_DECODE_ORDER`, decoder will not have any display delay, decoded frame data will get be free to display in same process call

This page is intentionally left blank

Appendix F

Support for Padding type

This version of decoder supports configurability to achieve desired padding process for non-multiple of 16 resolution video clips

It is recommended to utilize this feature only when the application is well aware of the nature of the encode bit-stream, as there may be some ambiguity can come for decoded YUV, below is explanation of the padding types used for decoder.

Desired padding process can be achieved by the application by setting IMPEG4VDEC_Params:: paddingMode.

Note:

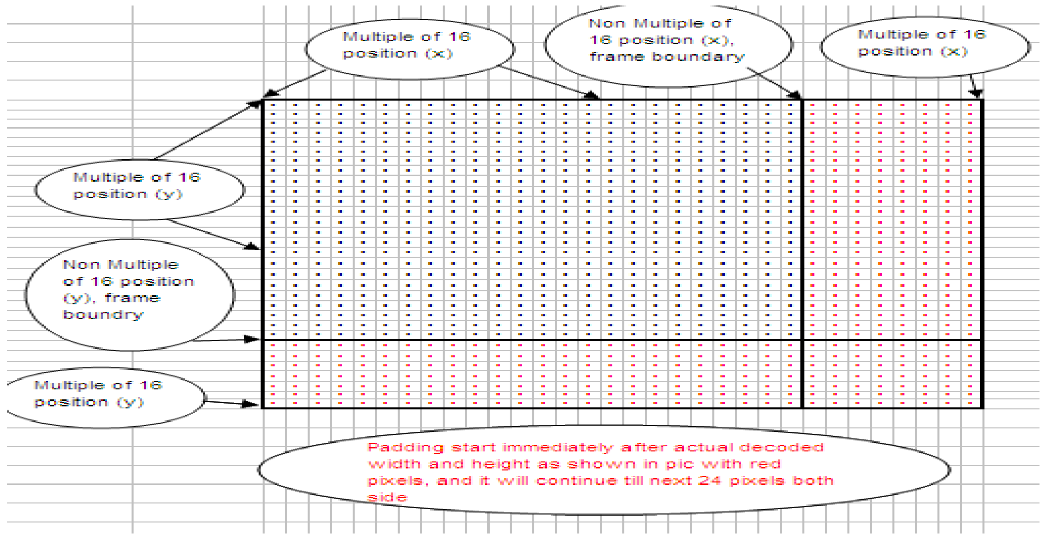
- ❑ MPEG4 Decoder supports two mode of padding type as specified in api file of the codec package
- ❑ `PAD_METHOD_DIVX`, decoder internally set padding type as Divx style of padding process for clip having resoltuin non multiple of 16.
- ❑ `PAD_METHOD_MPEG4`, decoder internally set padding type as mpeg4 style of padding process for clip having resoltuin non multiple of 16.

Below is the explanation of the both kind of padding process for non-multiple of 16 video clips.

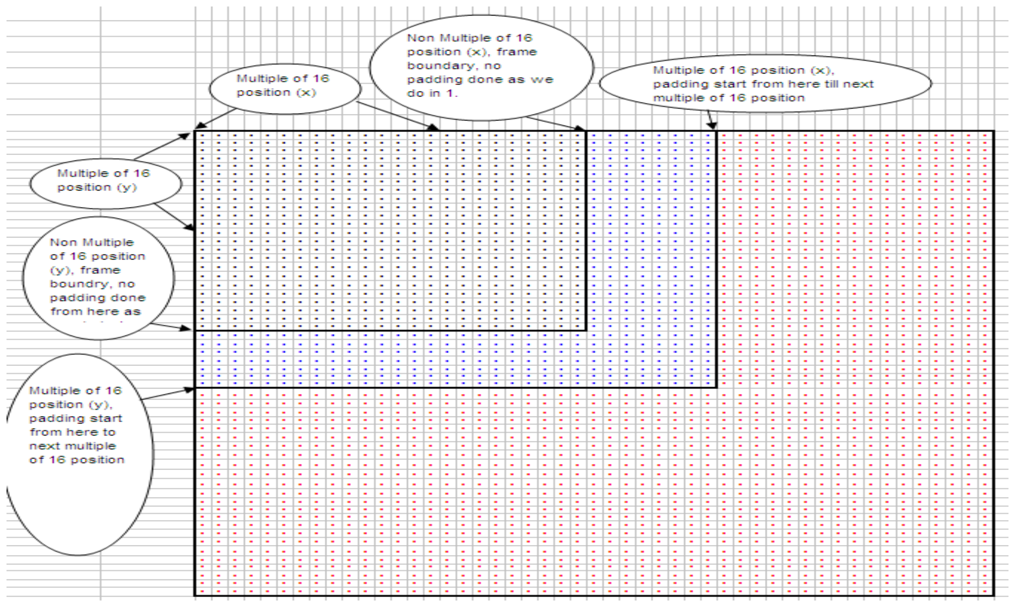
In MPEG4 Decoder, there is ambiguity to adopt the padding process (algorithm) for non-multiple of 16 resolutions video clips, and this ambiguity arise due to different interpretation of padding needs to be done by various encoder/decoder in case of non multiple of 16 resolution streams. There are two ways of interpretation of padding for non-multiple of 16 resolution streams.

- o Interpretation 1
- o Interpretation 2

Below pictorial view, will be more explanatory


Multiple of 16 position (x)
Non Multiple of 16 position (x), frame boundary
Multiple of 16 position (x)
Multiple of 16 position (y)
Non Multiple of 16 position (y), frame boundry
Multiple of 16 position (y)
Padding start immediately after actual decoded width and height as shown in pic with red pixels, and it will continue till next 24 pixels both side

First interpretation of the padding process defined in spec (also called as DIVX style of padding)


Multiple of 16 position (x)
Non Multiple of 16 position (x), frame boundary, no padding done as we do in 1.
Multiple of 16 position (x), padding start from here till next multiple of 16 position
Multiple of 16 position (y)
Non Multiple of 16 position (y), frame boundry, no padding done from here as
Multiple of 16 position (y), padding start from here to next multiple of 16 position

Second interpretation of the padding process defined in spec (also called as MPEG4 style of padding)

As defined above padding processes both interpretations are correct way of doing the padding for the non-multiple of 16-resolution stream, only concern comes when encoder follow one kind of padding and decoder follow other kind of padding. So to get the perfect output, both encoder and decoder shall follow identical interpretation of the standard, any mismatch in either side, may result in visual artifacts

This page is intentionally left blank

Appendix G

Support for Dynamic Change in Resolution

This version of decoder supports handling of change in resolution in as stream. Procedure to be as follows:

When decoder detects that, a change in resolution has occurred:

- Decoder shall send out the error code of IMPEG4D_ERR_PICSIZECHANGE
- Byte consumed value returned by codec shall not be inclusive of the VOL (mpeg4 video object layer) / VOP (H263 new frame sequence) beginning to new resolution.

When the application observes the error code of IMPEG4D_ERR_PICSIZECHANGE, it should take following steps:

- Flush out all frames locked inside decoder [these frames will be of older resolution]
- Perform control call to with XDM_GETSTATUS command to know the new resolution.
- Perform control call to with XDM_GETBUFINFO command to know the buffer requirement of the codec.
- Re-allocate the YUV buffers according to new resolution requirement
- Start performing process call again

Note:

- There is no need to perform XDM_RESET in the above flow.
- Above flow is same irrespective of whether the resolution increases or decreases.

This page is intentionally left blank

Appendix H

Support for Drop of frame

This version of decoder supports handling of drop of frame (I frame drop), meaning in case first frame of sequence is dropped then codec will be through error.

When decoder detects that, there is drop is frame meaning first frame not I frame then codec report error as follows:

- Decoder shall send out the error code of IMPEG4D_ERR_IFRAME_DROPPED
- Despite of reporting error as above codec will continue decoding of that frame data given to codec in process.

When the application observes the error code of IMPEG4D_ERR_IFRAME_DROPPED, it can take following steps, depend of application requirement:

- Application can ignore the decoded data and provide I frame data to decode to get correct decoded data from codec.
- Application can take decoded data and keep performing process, in this case codec will be giving visually wrong data unless it won't get new I Frame(As first I frame was dropped so codec will not be having correct reference data) .

Note:

- There is no need to perform XDM_RESET in the above flow.

This page is intentionally left blank

Appendix I

Support for Decoding only Intra frames using less memory

This version of decoder also supports decoding of only Intra frames from the stream. This feature is added to reduce the memory footprint requested by codec during create time. When decoder is created with this feature enabled, it shall request lesser memory during create time compared when this feature is disabled; and decode all the Intra frames from the streams. Please refer datasheet for gain in memory obtained with this feature.

Important points regarding this feature –

- If this feature is enabled for a stream with P/B frames, the decoder searches for next frame's start code till it finds a Intra frame in the bytes given to decoder. It is advisable to use this feature for streams with all Intra frames.
- The output buffer given for the decoder for a particular process call is displayed, if a Intra frame is decoded, and freed for that process call only.
- Decoder does not give any output buffers when it encounters P/B frames.
- XDM_FLUSH shouldn't be called when this feature is enabled. But if it is called then decoder doesn't give any output buffers.

Note:

MPEG4 Decoder supports two values for decodeOnlyIntraFrames parameter in create time structure for this feature -

- ❑ IMPEG4VDEC_DECODE_ONLY_I_FRAMES_DISABLE, during create time, decoder requests memory required to decode all frame types.
- ❑ IMPEG4VDEC_DECODE_ONLY_I_FRAMES_ENABLE, decoder requests lesser memory required just to decode only I frames

This page is intentionally left blank