MPEG2 Main Profile Decoder on DM6467

User Guide

Literature Number: SPRUFE1
June 2008

TEXAS INSTRUMENTS

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

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

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

Read This First

About This Manual

This document describes how to install and work with Texas Instruments' (TI) MPEG2 Main Profile Decoder implementation on the DM6467 platform. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

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

Intended Audience

This document is intended for system engineers who want to integrate TI's codecs with other software to build a multimedia system based on the DM6467 platform.

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

How to Use This Manual

This document includes the following chapters:

- **Chapter 1 - Introduction**, 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.

Related Documentation From Texas Instruments

The following documents describe TMS320 devices and related support tools. 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.
- *TMS320c64x+ Megamodule* (literature number SPRAA68) describes the enhancements made to the internal memory and describes the new features which have been added to support the internal memory architecture's performance and protection.
- *TMS320c64x+ Megamodule* (literature number SPRAA68) describes the enhancements made to the internal memory and describes the new features which have been added to support the internal memory architecture's performance and protection.
- *TMS320C64x+ DSP Megamodule Reference Guide* (literature number SPRU871) describes the C64x+ megamodule peripherals.
- *TMS320C64x to TMS320C64x+ CPU Migration Guide* (literature number SPRAA84) describes migration from the Texas Instruments TMS320C64x™ digital signal processor (DSP) to the TMS320C64x+™ DSP.
- *TMS320C6000 Optimizing Compiler v 6.0 Beta User's Guide* (literature number SPRU187N) explains how to use compiler tools such as compiler, assembly optimizer, standalone simulator, library-build utility, and C++ name demangler.
- *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (literature number SPRU732) describes the CPU architecture, pipeline, instruction set, and interrupts of the C64x and C64x+ DSPs.
- *DaVinci Technology - Digital Video Innovation Product Bulletin (Rev. A)* (literature number SPRT378A)
- *The DaVinci Effect: Achieving Digital Video Without Complexity White Paper* (literature number SPRY079)

- *DaVinci Benchmarks Product Bulletin* (literature number SPRT379)
- *DaVinci Technology for Digital Video White Paper* (literature number SPRY067)
- *The Future of Digital Video White Paper* (literature number SPRY066)

Related Documentation

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

- *ISO/IEC 13818-2: 1995 (E) Rec. MPEG2 (E) ITU-T Recommendation*

Abbreviations

The following abbreviations are used in this document.

Table 1-1. List of Abbreviations

Abbreviation	Description
1080p	1920 x 1088 resolution in progressive scan.
BIOS	TI's simple RTOS for DSPs
CSL	Chip Support Library
D1	720x480 or 720x576 resolutions in progressive scan
DCT	Discrete Cosine Transform
DMA	Direct Memory Access
DMAN	DMA Manager
HDTV	High Definition Television
IRES	Interface standard to request and receive handles to resources
MB	Macro Block
MPEG	Motion Pictures Expert Group
MV	Motion Vector
NTSC	National Television Standards Committee
RMAN	Resource Manager
RTOS	Real Time Operating System

Abbreviation	Description
VGA	Video Graphics Array
XDAIS	eXpressDSP Algorithm Interface Standard
XDM	eXpressDSP Digital Media
YUV	Color space in luminance and chrominance form

Text Conventions

The following conventions are used in this document:

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

Product Support

When contacting TI for support on this codec, quote the product name (MPEG2 Decoder on DM6467) and version number. The version number of the codec is included in the title of the release notes that accompanies this codec.

Trademarks

Contents

Figures

This page is intentionally left blank

Tables

This page is intentionally left blank

Chapter 1

Introduction

This chapter provides a brief introduction to XDAIS and XDM. It also provides an overview of TI's implementation of the MPEG2 Main Profile Decoder on the DM6467 platform and its supported features.

Topic **Page**

1.1 Overview of XDAIS and XDM

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

1.1.1 XDAIS Overview

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

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

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

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

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

1.1.2 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 MPEG2) in your system. To enable easy integration with the client application, it is important that all codecs with similar functionality use similar APIs. XDM was primarily defined as an extension to XDAIS to ensure uniformity across different classes of codecs (for example audio, video, image, and speech).

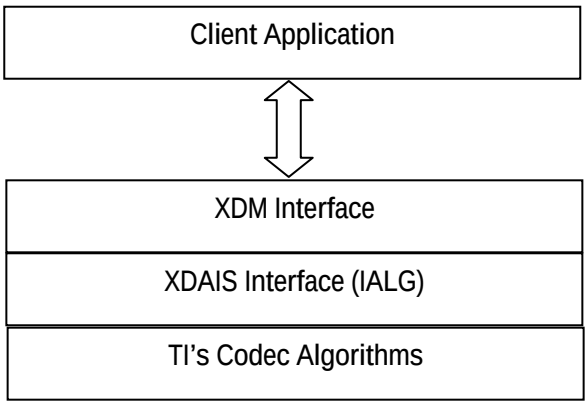
The XDM standard defines the following two APIs:

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

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

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

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



As depicted in the figure, XDM is an extension to XDAIS 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 MPEG2 video decoder, then you can easily replace MPEG2 with another XDM-compliant video decoder.

1.2 Overview of MPEG2 Main Profile Decoder

The MPEG2 video standard specifies the decompression and coded representation for entertainment-quality digital video. It is widely used in different digital video systems, including DTV (Digital Television), DVB (Digital Video Broadcast), DSS (Direct Satellite System), and DVD (Digital Versatile Disc). The MPEG2 video decoder plays an important role in consumer electronics like DVD players, set-top boxes, and DSS units.

Figure 1-1 depicts the flow diagram of the MPEG2 Decoder.

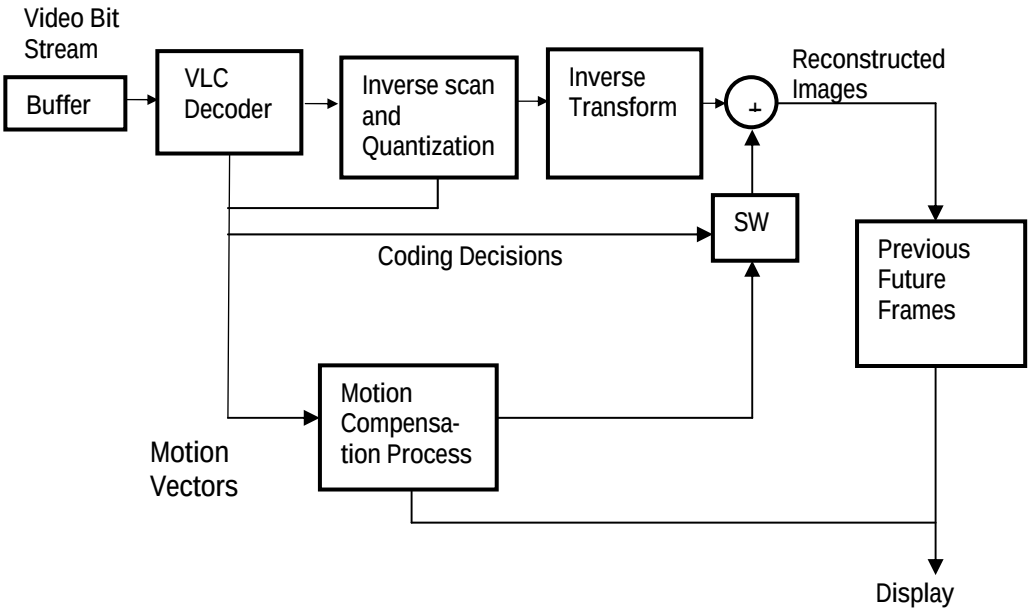


Figure 1-1. Flow Diagram of MPEG2 Decoder.

1) Simple Profile:

   - Allows the use of I–frames(Intra-coded) and P–frames(predicted).

2) Main Profile:

   - Allows the use of I–frames (Intra-coded), P–frames (predicted) and B–frames(bidirectional predicted).

From this point onwards, all references to MPEG2 Decoder means MPEG2 Main Profile Decoder only.

1.3 Supported Services and Features

This user guide accompanies TI's implementation of MPEG2 Main Profile Decoder on the DM6467 platform. This version of the codec has following supported features:

- eXpressDSP Digital Media (XDM 1.2 IVIDDEC2) complaint
- Supports all I, P, and B frame decoding
- Supports both progressive and interlaced
- Outputs are available in YUV 420 interleaved formats (Y in one plane and U and V data interleaved to form the other plane)
- Supports frame based decoding with frame size being multiples of 16
- Supports Simple Profile MPEG2 decoding
- Supports Main Profile MPEG2 decoding
- Supports DMA based framework
- Supports use of C64x+ and ARM968 of HD-VICP0 and HDVICP1
- Supports interrupt based communication between processors

This page is intentionally left blank

Chapter 2

Installation Overview

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

Topic **Page**

2.1 System Requirements

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

2.1.1 Hardware

This codec has been built and tested on DM6467 EVM only.

2.1.2 Software

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

- **Development Environment:** This project is developed using Code Composer Studio version 3.3.49. It has been tested with DM6467 EVM.
- **Code Generation Tools:** This project is compiled, assembled, archived, and linked using the code generation tools version 6.0.8.

2.2 Installing the Component

The codec component is released as a compressed archive. To install the codec, extract the contents of the zip file onto your local hard disk. The zip file extraction creates a top-level directory called 200_V_MPEG2_D_1_00 under which another directory named DM6467_MP_001 is created.

Figure 2-1 shows the sub-directories created in the DM6467_MP_001 directory.

Note:

The source folders under Mpeg2Decoder (`AlgSrc, Arm968`) is not present in the case of a library based (object code) release.

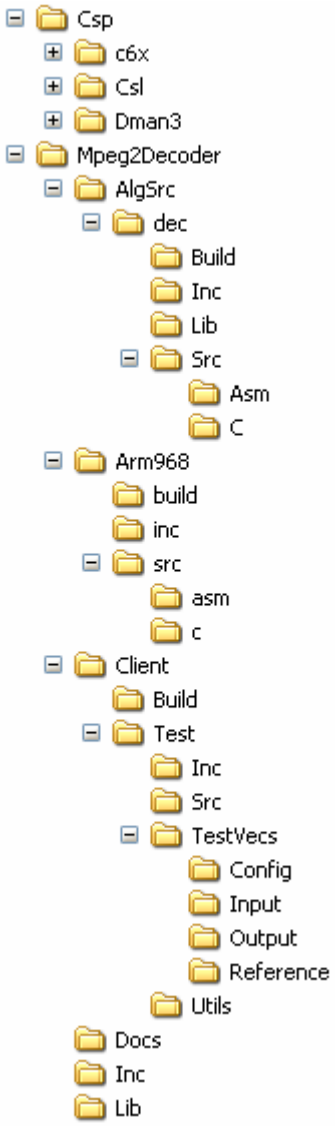


Figure 2-1. Component Directory Structure

Note:

If you are installing an evaluation version of this codec, the directory name will be 200E_V_ MPEG2_D_1_00.

Table 2-1. Component Directories

Sub-Directory	Description
Csp\Csl	Contains CSL files.
Csp\C6x	Contains CSL files
Csp\Dman3	Contain DMAN3 related files.
\Mpeg2Decoder\Inc	Contains XDM related header files which allow interface to the codec library
\Mpeg2Decoder \Lib	Contains generated algorithm library file .
\Mpeg2Decoder \Docs	Contains user guide and datasheet
\Mpeg2Decoder \Client\Build	Contains the sample test application project (.pjt) file
\Mpeg2Decoder \Client\Test\Src	Contains application C files
\Mpeg2Decoder \Client\Test\Inc	Contains header files needed for the application code
\Mpeg2Decoder \Client\Test\Utils	Folder to store basic utilities like file compare and YUV display executables.
\Mpeg2Decoder \Client\Test\TestVecs\Input	Contains input test vectors
\Mpeg2Decoder \Client\Test\TestVecs\Output	Contains output generated by the codec
\Mpeg2Decoder \Client\Test\TestVecs\Reference	Contains read-only reference output to be used for bit-compliance checking
\Mpeg2Decoder \Client\Test\TestVecs\Config	Contains configuration parameter file
\Mpeg2Decoder \AlgSrc\dec\Src\C	Contains algorithm source C files
\Mpeg2Decoder \AlgSrc\dec\Src\Asm	Contains algorithm source Asm files
\Mpeg2Decoder \AlgSrc\dec\Lib	Used earlier to contain algorithm library file.
\Mpeg2Decoder \AlgSrc\dec\Inc	Contains algorithm header files
\Mpeg2Decoder \AlgSrc\dec\Build	Contains the algorithm application project (.pjt) file
\Mpeg2Decoder \Arm968\src	Contains the ARM968 source files
\Mpeg2Decoder \Arm968\inc	Contains the header files for Arm968 project

Sub-Directory	Description
\Mpeg2Decoder \Arm968\build	Contains the ARM968 project file

2.3 Before Building the Sample Test Application

This codec is accompanied by a sample test application. To run the sample test application, you need DSP/BIOS, TI Codec Engine (CE) and HDVICP API. This version of the codec has been validated with DSP/BIOS version 5.31, Codec Engine (CE) version 2.10.01 and HDVICP API version 1.01.000.

2.3.1 Installing DSP/BIOS

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

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

Install DSP/BIOS at the same location where you have installed Code Composer Studio. For example:<install directory>\CCStudio_v3.3

Set system environment variable `BIOS_INSTALL_DIR` to point to the location where the <bios_directory> is present.

The sample test application may use the following DSP/BIOS files:

- Header file, bcache.h available in the <install directory>\CCStudio_v3.3\<bios_directory>\packages \ti\bios\include directory.
- Library file, biosDM420.a64P available in the <install directory>\CCStudio_v3.3\<bios_directory>\packages\ti\bios\lib directory.

2.3.2 Installing Codec Engine (CE)

Download CE version 2.10.01 or newer from TI's external website:

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

The codec uses framework components and XDIAS version that are a part of CE 2.10.01 or newer.

1) Extract the CE zip file to the same location where you have installed Code Composer Studio. For example: <install directory>\CCStudio_v3.3.

   The test application uses the following RMAN files:

   - Library file, rmand.a64P available in the <install directory>\ CCStudio_v3.3\<ce_directory>\cetools\packages\ti\sdo\fc\rman directory.
   - Header file, rman.h available in the <install directory>\CCStudio_v3.3 \<ce_directory>\cetools\packages\ti\sdo\fc\rman directory.

- o Header file, ires.h available in the <install directory>\CCStudio_v3.3\<ce_directory>\cetools\packages\ti\xdais directory.

2) Set a system environment variable named `FC_INSTALL_DIR` pointing to <install directory>\CCStudio_v3.3\<ce_directory>\cetools.

3) Set a system environment variable named `XDAIS_INSTALL_DIR` pointing to <install directory>\CCStudio_v3.3\<ce_directory>\cetools\packages\ti\xdais.

4) Set a system environment variable `EDMA3LLD_INSTALL_DIR` pointing to <install directory>\CCStudio_v3.3\<ce_directory>\cetools\packages\ti\sdo\edma3.

2.3.3 Installing HDVICP API

1) Extract the HDVICP API zip file to the same location where the Code Composer Studio is installed. For example: <install directory>\CCStudio_v3.3.

2) Set a system environment variable named HDVICP_API pointing to <install directory>\CCStudio_v3.3\<hdvicp>\200_V_HDVICP_X_1_01\DM6467_X_001\Hdvicp_api

2.4 Building and Running the Sample Test Application

This codec is accompanied by a sample test application. The application will run in TI's Code Composer Studio development environment. To build and run the sample application in Code Composer Studio, follow these steps:

1) Extract the .zip file.

2) Verify that you have installed TI's Code Composer Studio version 3.3 with SR 3.0 and code generation tools version 6.0.8. Start the Code Composer Studio to view the Parallel Debug Manager (PDM) window.

3) In the PDM window, open the window by double clicking on ARM926, load the GEL file davincihd_arm.gel and click on **Debug >Connect**.

4) In the PDM window, open the window by double clicking on C6400PLUS, load the GEL file davincihd_dsp.gel and click on **Debug >Connect**.

5) Open the test application project file in C6400PLUS window, mpeg2vdec_ti_c64xplustestapp.pjt in Code Composer Studio. This file is available in the \Mpeg2Decoder\Client\Build sub-directory.

6) Select **Project > Build** to build the sample test application. This will also build the dependent project mpeg2vdec_ti_c64xplus.pjt stored at location Mpeg2Decoder\AlgSrc\Src\Build and create the final executable file, mpeg2vdec_ti_c64xplustestapp.out at \Mpeg2Decoder\Client\Build\Out sub-directory.

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

8) Select **Debug > Run** to execute the sample test application on DSP.

   The sample test application takes the input files stored in the \Mpeg2Decoder\Client\Test\TestVecs\Input sub-directory, runs the codec, and dumps the output files at \Mpeg2Decoder\Client\Test\TestVecs\Output sub-directory. The user can use the reference files stored in \Mpeg2Decoder\Client\Test\TestVecs\Reference sub-directory to compare and check for compliance.

2.5 Configuration Files

This codec is shipped along with two configuration files:

- Generic configuration file (Testvecs.cfg) – specifies input and reference files for the sample test application.
- Decoder configuration file (Testparams.cfg) – specifies the configuration parameters required for the Decoder.

2.5.1 Generic Configuration File

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

The format of the Testvecs.cfg file is:

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

where:

- `X` may be set as:
  - 1 - for compliance checking, no output file is created (This option is not supported currently)
  - 0 - for writing the output to the output file

  The default setting of Testvecs.cfg file is for compliance checking.
- `Config` is the Decoder configuration file. For details, see Section 2.5.2.
- `Input` is the input file name with complete path
- `Output/Reference` is the output file name (if `X` is 0) or reference file name (if `X` is 1).

A sample Testvecs.cfg file is as shown:

```
0
testparams.cfg   /* Test Parameters File with path */
test.m2v         /* Input File with path */
OutFrame.yuv     /* Output File */

1
testparams.cfg   /* Test Parameters File with path */
test.m2v         /* Input File with path */
OutFrame.yuv     /* Reference File */
```

2.5.2 Decoder Configuration File

The decoder configuration file, Testparams.cfg contains the configuration parameters required for the decoder. The Testparams.cfg file is available in the \Client\Test\TestVecs\Config sub-directory.

A sample Testparams.cfg file is as shown:

```
# New Input File Format is as follows
# <ParameterName> = <ParameterValue> # Comment#
############################################################
# Parameters
############################################################
ImageWidth    =  128  # Image width in Pels, must be multi-
ple of 16
ImageHeight   =  128  # Image height in Pels, must be multi-
ple of 16
ChromaFormat  =  1    # 1 => XMI_YUV_420P, 3 =>
XMI_YUV_422IBE, 4 => XMI_YUV_422ILE
FramesToDecode = 3    # Number of frames to be coded
```

Any field in the `IVIDDEC2_Params` structure (see section 4.2.1.8) can be set in the Testparams.cfg file using the syntax shown above. If you specify additional fields in the Testparams.cfg file, ensure to modify the test application appropriately to handle these fields.

The following parameters are specified in this file, with each parameter on a new line:

- Maximum video width
- Maximum video height
- Output chroma format (Only 420 chroma format is supported : 1)
- Number of frames to decode

2.6 Standards Conformance and User-Defined Inputs

To check the conformance of the codec for the input files shipped along with the codec, compare the outputs dumped in the \Mpeg2Decoder\Client\Test\TestVecs\Output folder with the corresponding reference files present in the \Mpeg2Decoder\Client\Test\TestVecs\Reference folder.

For the input file to have passed, the maximum difference for each pixel comparison of the files should not exceed +/- 1.

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

1) Copy the input files to the \Mpeg2Decoder\Client\Test\TestVecs\Input sub-directory.

2) Copy the reference files to the \Mpeg2Decoder\Client\Test\TestVecs\Reference sub-directory.

3) Edit the configuration file, Testvecs.cfg available in the \Mpeg2Decoder\Client\Test\TestVecs\Config sub-directory. For details on the format of the Testvecs.cfg file, see Section 2.5.2.

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

   - "PASS/FAIL" (if X is 1): Not supported in this version of MPEG2 Decoder.
   - "Displaying Frame" (if X is 0)

If you have chosen to write to an output file (X is 0), you may use any standard file comparison utility to compare the codec output with the reference output and check for conformance. If the maximum pixel difference between the output and reference files does not exceed +/- 1, the stream is supposed to have passed.

2.7 Uninstalling the Component

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

2.8 Evaluation Version

If you are using an evaluation version of this codec, there will be a limit of decoding 54000 frames.

This page is intentionally left blank

Chapter 3

Sample Usage

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

Topic **Page**

3.1 Overview of the Test Application

The test application is an `IVIDDEC2` base class of the MPEG2 Main Profile Decoder library. The main test application files are testappdecoder_mpeg2.c and testappdecoder.h. These files are available in the \Mpeg2Decoder\Client\Test\Src and \Mpeg2Decoder\Client\Test\Inc subdirectories respectively.

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

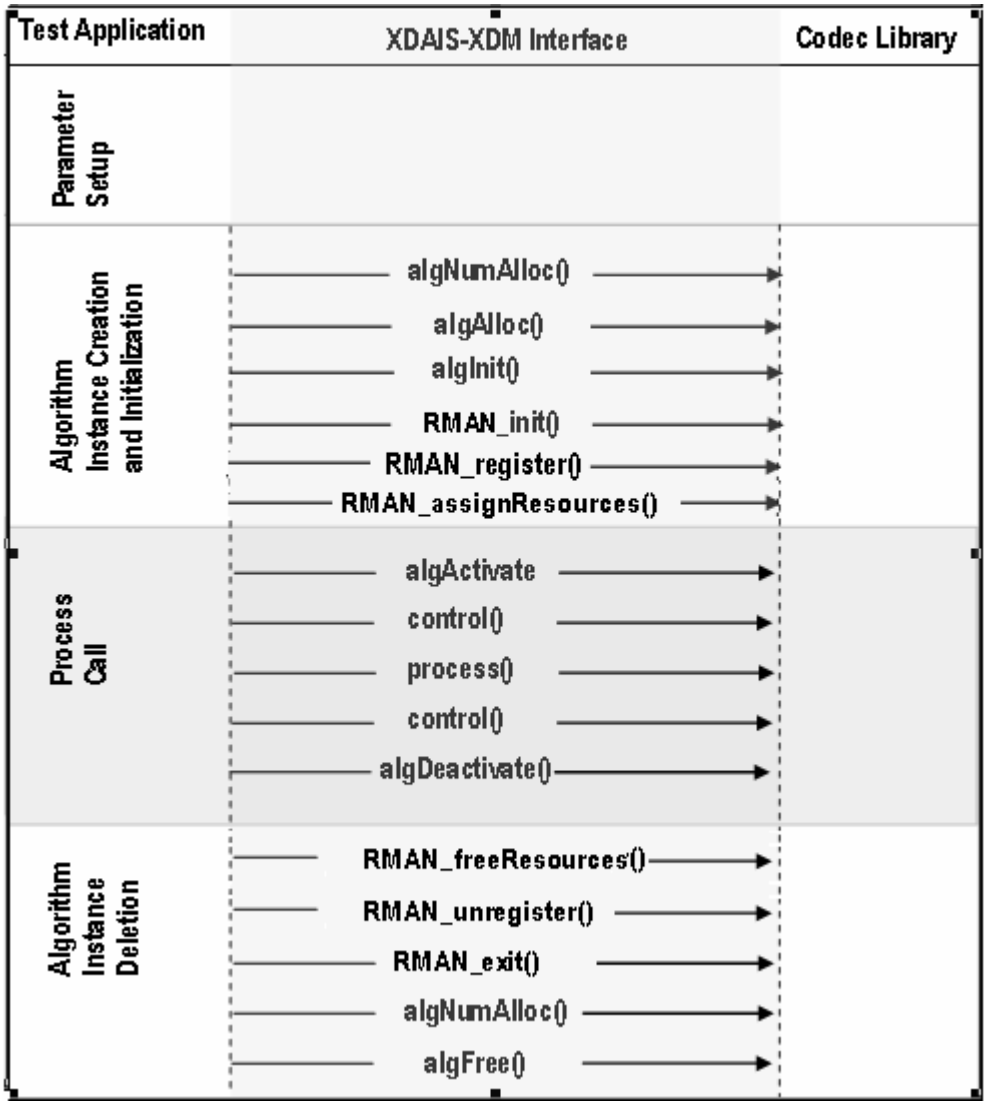


Figure 3-1. Test Application Sample Implementation

The test application is divided into four logical blocks:

1) Parameter setup
2) Algorithm instance creation and initialization
3) Process call
4) 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 `IVIDDEC2_Params` structure based on the values it reads from the Testparams.cfg file.
4) Reads the input bit-stream into the application input buffer.

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

3.1.2 Algorithm Instance Creation and Initialization

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

1) `algNumAlloc()` - To query the algorithm about the number of memory records it requires.
2) `algAlloc()` - To query the algorithm about the memory requirement to be filled in the memory records.
3) `algInit()` - To initialize the algorithm with the memory structures provided by the application.

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

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

1) `RMAN_init()` - To initialize the RMAN module.

2) `RMAN_register()` - To register the HDVICP protocol/resource manager with the generic resource manager.

3) `RMAN_assignResources` - To register resources to the algorithm as requested HDVICP protocol/resource manager with the generic resource manager.

Note:

RMAN function implementations are provided in rmand.a64P library.

3.1.3 Process Call

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

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

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

3) Implements the process call based on the non-blocking mode of operation. The behavior of the algorithm can be controlled using various dynamic parameters (see Section 4.2.1.6). The inputs to the `process()` functions are input and output buffer descriptors, pointer to the `IVIDDEC2_InArgs` and `IVIDDEC2_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 for the end of frame encode/decode, the application should release the semaphore and resume the video task, which will do any book-keeping operations by the codec and updating the output parameters structure - `IVIDDEC2_OutArgs`.

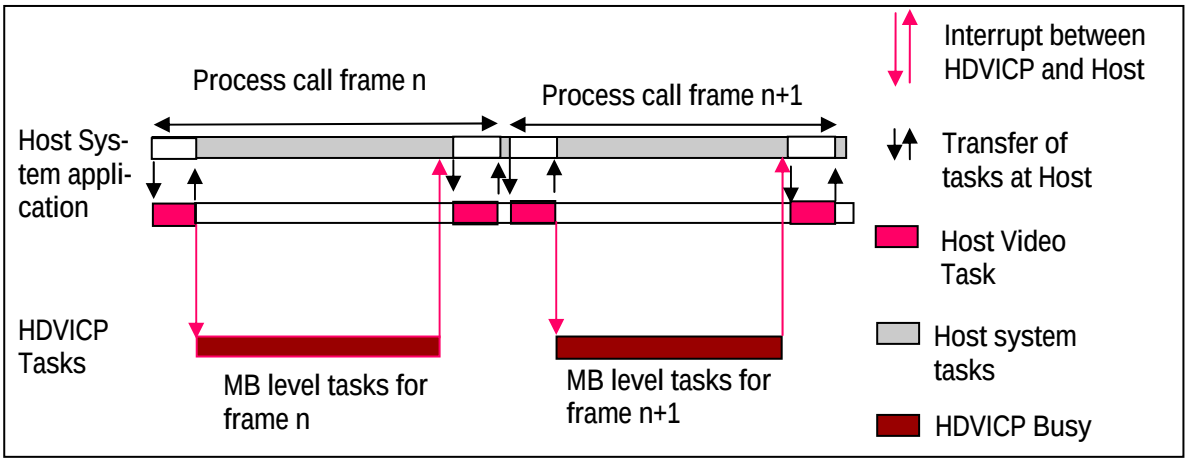


Figure 3-2. Process Call with Host Release

Note:

- The process call returns control to the application after the initial setup related tasks are completed
- Application can schedule a different task to use the freed up Host resource
- All service requests from HDVICP handled through interrupts
- Application resumes the suspended process call after last service request for HDVICP 0/1 has been handled
- Application can now complete concluding portions of the process call and return

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

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

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

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

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

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

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

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

3.1.4 Algorithm Instance Deletion

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

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

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

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

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

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

2) `RMAN_unregister()` - To un-register the HDVICP protocol/resource manager with the generic resource manager.

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

3.2 Frame Buffer Management by Application

3.2.1 Frame Buffer Input and Output

With the new XDM 1.0, decoder does not ask for frame buffer at the time of `alg_create()`. It uses buffer from `XDM1_BufDesc *outBufs`, which the decoder gets during each decode process call. Hence, there is no distinction between reference and display buffers. The framework needs to ensure that it does not overwrite to buffers, which are locked by the codec.

```
mp2VDEC_create();
mp2VDEC_control(XDM_GETBUFINFO); /* Returns default PAL D1
size */
do{
mp2VDEC_decode(); //call the decode API
mp2VDEC_control(XDM_GETBUFINFO); /* updates the memory re-
quired 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 also re-use the extra buffer space of the 1st frame, if the above control call returns a smaller size than that is returned in the first call.

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 the record of buffer given to algorithm and released by algorithm. The figure below explains the frame pointer usage.

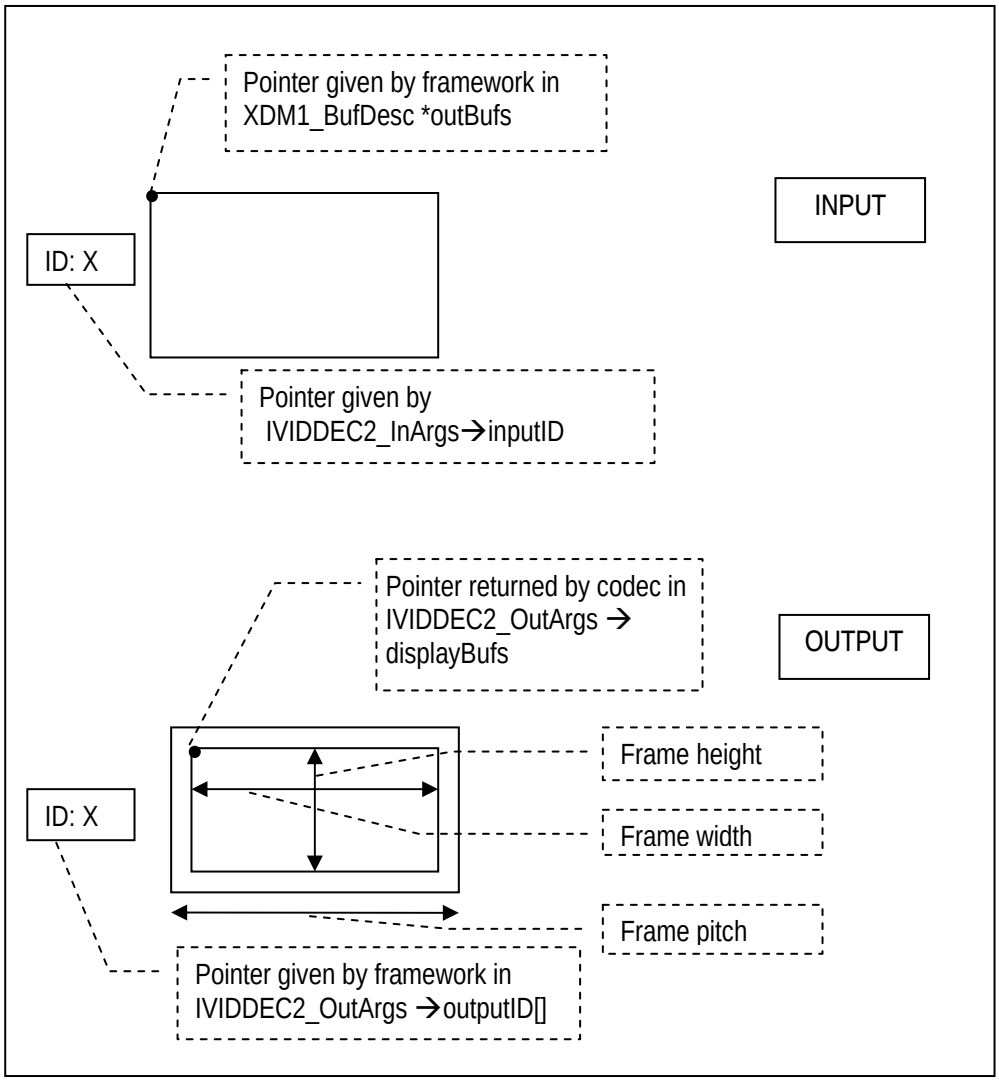


Figure 3-3. Frame Buffer Pointer Implementation

Note:

- Frame pointer returned by codec in `displayBufs` points to the actual start location of picture
- Frame height and width is the actual height and width (after removing cropping and padded width)
- Frame pitch indicates the offset between the pixels at the same horizontal coordinate on two consecutive lines

As explained above, buffer pointer cannot be used as a unique identifier to keep a record of frame buffers. Any buffer given to the algorithm should be considered locked by the algorithm, unless the buffer is returned to the application through `IVIDDEC2_OutArgs->freeBufID[]`.

Note:

BufferID returned in IVIDDEC2_OutArgs ->outputID[] is for display purpose. Application should not consider it free, unless it comes as part of IVIDDEC2 OutArgs->freeBufID[]

3.2.2 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.

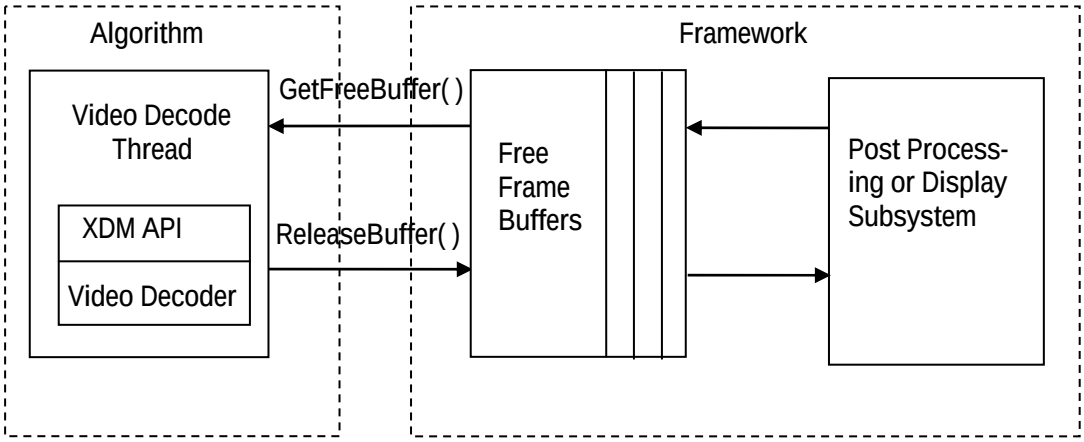


Figure 3-4. Interaction of Frame Buffers Between Application and Framework

The sample application also provides a prototype for managing frame buffers. It implements the following functions, which are defined in file 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 one frame of memory data for the first process call output buffers depending on the maximum width and maximum height supplied in params.
- BUFFMGR_ReInit() - BUFFMGR_ReInit function allocates the remaining luma and chroma buffers required by the stream based on the actual picture width and height obtained after first process call.
- BUFFMGR_GetFreeBuffer() - BUFFMGR_GetFreeBuffer function searches for a free buffer in global buffer array and returns the address of that element. Incase, if none of the elements are free then it returns NULL.
- BUFFMGR_ReleaseBuffer() - BUFFMGR_ReleaseBuffer function takes an array of buffer-IDs, which are released by the test-app. Zero is not a valid buffer ID. Hence, this function keeps moving 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.

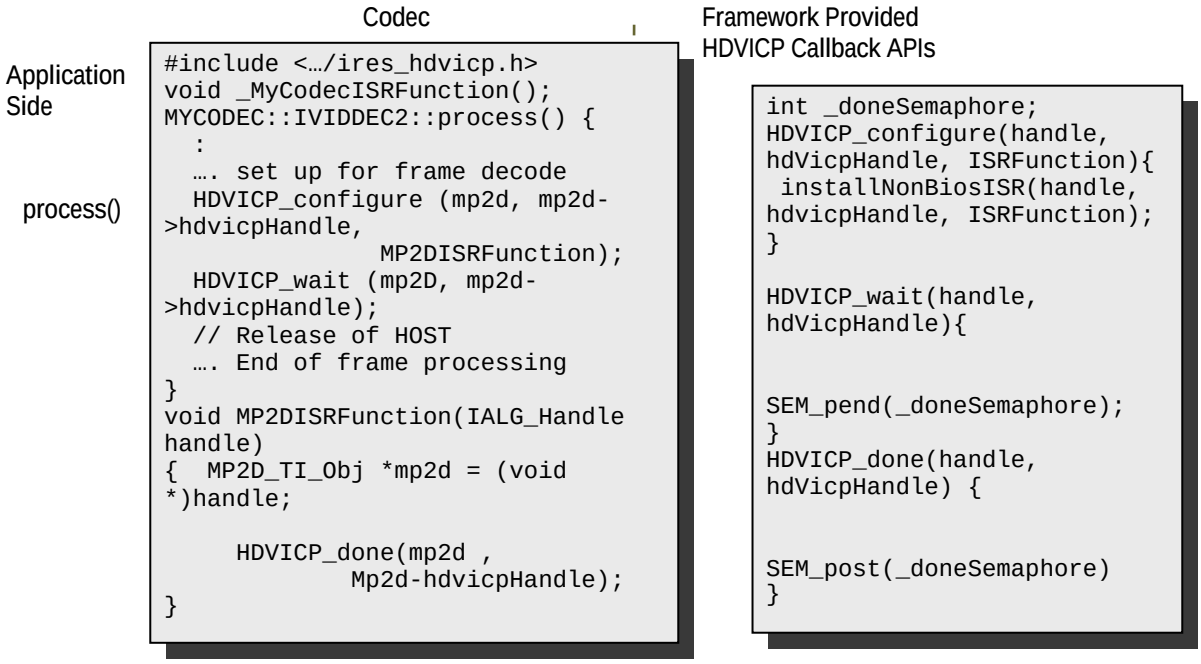


Figure 3-5. Interaction Between Application and Codec.

Note:

- Process call architecture shares 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`) are also outside the codec.
- Codec is OS independent.

The functions to be implemented by the application are:

1) `HDVICP_configure(IALG_Handle handle, void *hdvicpHandle, void (*ISRfunctionptr)(IALG_Handle handle))`

   This function is called by the algorithm to register its ISR function, which the application needs to call when it receives, interrupts pertaining to video task.

2) `HDVICP_wait (void *hdvicpHandle)`

   This function is called by the algorithm to move the video task to SEM-pend state.

3) `HDVICP_done (void *hdvicpHandle)`

   This function is called by the algorithm to release the video task from SEM-pend state.

   In the sample test application, these functions are implemented in hdvicp_framework.c file. The application can implement it in their own way considering the underlying system.

3.4 Sample Test Application

The test application exercises the `IVIDDEC2` base class of the Mpeg2 Decoder.

Table 3-1. Process () Implementation.

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

  BUFFMGR_Init();


  TestApp_SetInitParams(&params.viddecParams);

  HDVICP_initHandle(&hdvicpObj);
  /*---------------- Decoder creation -----------------*/
  handle = (IALG_Handle) mp2VDEC_create();

      /* Get Buffer information              */
  mp2VDEC_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 con-
trol() */
mp2VDEC_control(handle, XDM_SETPARAMS);
```

```
/*-----------------------------------------------*/
/* Start the process : To start decoding a frame      */
/* This will always follow a mp2VDEC_decode_end call  */
/*-----------------------------------------------*/
      retVal = mp2VDEC_decode
                  (
                    handle,
                    (XDM1_BufDesc *)&inputBufDesc,
                    (XDM_BufDesc *)&outputBufDesc,
                    (IVIDDEC2_InArgs *)&inArgs,
                    (IVIDDEC2_OutArgs *)&outArgs
                  );

/* Get the status of the decoder using control call */
       mp2VDEC_control(handle, Imp2VDEC_GETSTATUS);

/* Get Buffer information :      */
       mp2VDEC_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.

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
	IVIDEO_MBAFF_IDR_FRAME	Intra coded MBAFF frame that can be used for refreshing video content.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IVIDEO_FRAMETYPE_DEFAULT	Default value is set to IVIDEO_I_FRAME
IVIDEO_ContentType	IVIDEO_CONTENTTYPE_NA	Content type is not applicable
	IVIDEO_PROGRESSIVE	Progressive video content
	IVIDEO_PROGRESSIVE_FRAME	Progressive video content
	IVIDEO_INTERLACED	Interlaced video content.
	IVIDEO_INTERLACED_FRAME	Interlaced video content.
	IVIDEO_INTERLACED_TOPFIELD	Interlaced picture - top field
	IVIDEO_INTERLACED_BOTTOMFIELD	Interlaced picture - bottom field
	IVIDEO_CONTENTTYPE_DEFAULT	Default value is set to IVIDEO_PROGRESSIVE
IVIDEO_FrameSkip	IVIDEO_NO_SKIP	Do not skip the current frame. This is the default value.
	IVIDEO_SKIP_P	Skip forward inter coded frame. Not supported in this version of MPEG2 decoder.
	IVIDEO_SKIP_B	Skip bi-directional inter coded frame. Not supported in this version of MPEG2decoder.
	IVIDEO_SKIP_I	Skip intra coded frame. Not supported in this version of MPEG2 decoder.
	IVIDEO_SKIP_IP	Skip I and P frame/field(s)
	IVIDEO_SKIP_IB	Skip I and B frame/field(s).
	IVIDEO_SKIP_PB	Skip P and B frame/field(s).
	IVIDEO_SKIP_IPB	Skip I/P/B/BI frames
	IVIDEO_SKIP_IDR	Skip IDR Frame
	IVIDEO_SKIP_DEAFAULT	Default value is set to IVIDEO_NO_SKIP
XDM_DataFormat	XDM_BYTE	Big endian stream
	XDM_LE_16	16-bit little endian stream. Not applicable for MPEG2 decoder.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_LE_32	32-bit little endian stream. Not applicable for MPEG2 decoder.
	XDM_LE_64	64 bit little endian stream. Not applicable for MPEG2 decoder.
	XDM_BE_16	16 bit big endian stream. Not applicable for MPEG2 decoder.
	XDM_BE_32	32 bit big endian stream. Not applicable for MPEG2 decoder.
	XDM_BE_64	64 bit big endian stream. Not applicable for MPEG2 decoder.
XDM_ChromaFormat	XDM_CHROMA_NA	Chroma Format not applicable.
	XDM_YUV_420P	YUV 4:2:0 planar
	XDM_YUV_422P	YUV 4:2:2 planar. Not applicable for MPEG2 decoder.
	XDM_YUV_422IBE	YUV 4:2:2 interleaved (big endian). Not applicable for MPEG2 decoder.
	XDM_YUV_422ILE	YUV 4:2:2 interleaved (little endian)
	XDM_YUV_444P	YUV 4:4:4 planar. Not applicable for MPEG2 decoder.
	XDM_YUV_411P	YUV 4:1:1 planar. Not applicable for MPEG2 decoder.
	XDM_GRAY	Gray format. Not applicable for MPEG2 decoder.
	XDM_RGB	RGB color format. Not applicable for MPEG2 decoder
	XDM_CHROMAFORMAT_DEFAULT	Default value is set to XDM_YUV_422ILE
XDM_CmdId	XDM_GETSTATUS	Query algorithm instance to fill Status structure
	XDM_SETPARAMS	Set run-time dynamic parameters through the DynamicParams structure
	XDM_RESET	Reset the algorithm
	XDM_SETDEFAULT	Initialize all fields in Params structure to default values specified in the library

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	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 is returned in the @c data field of the respective _Status structure. Not supported in this version of MPEG2 Decoder.
XDM_AccessMode	XDM_ACCESSMODE_READ	The algorithm read from the buffer using the CPU.
	XDM_ACCESSMODE_WRITE	The algorithm wrote from the buffer using the CPU.
XDM_ErrorBit	XDM_PARAMSCHANGE	Bit 8 ❑ 1 - Sequence parameters change ❑ 0 - Ignore
	XDM_APPLIEDCONCEALMENT	Bit 9 ❑ 1 - Applied concealment ❑ 0 - Ignore
	XDM_INSUFFICIENTDATA	Bit 10 ❑ 1 - Insufficient data ❑ 0 - Ignore
	XDM_CORRUPTEDDATA	Bit 11 ❑ 1 - Data problem/corruption ❑ 0 - Ignore
	XDM_CORRUPTEDHEADER	Bit 12 ❑ 1 - Header problem/corruption ❑ 0 - Ignore
	XDM_UNSUPPORTEDINPUT	Bit 13 ❑ 1 - Unsupported feature/parameter in input ❑ 0 - Ignore
	XDM_UNSUPPORTEDPARAM	Bit 14 ❑ 1 - Unsupported input parameter or configuration ❑ 0 - Ignore
	XDM_FATALERROR	Bit 15 ❑ 1 - Fatal error (stop encoding) ❑ 0 - Recoverable error

Note:

The remaining bits that are not mentioned in `XDM_ErrorBit` are interpreted as:

- Bit 16-32:Reserved
- Bit 8: Reserved
- Bit 0-7:Codec and implementation specific
- Only Bit 0 is presently used by algorithm to signal junk data error. This bit is set and algorithm comes out with no output frames if input data does not contain start code for any of the following :

  Sequence Header

  Group of Pictures header

  Picture header

  Sequence End code

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

4.2 Data Structures

This section describes the XDM defined data structures, which are common across codec classes. These XDM data structures can be extended to define any implementation specific parameters for a codec component.

4.2.1 Common XDM Data Structures

This section includes the following common XDM data structures:

- `XDM_BufDesc`
- `XDM1_BufDesc`
- `XDM_SingleBufDesc`
- `XDM1_SingleBufDesc`
- `XDM_AlgBufInfo`
- `IVIDEO1_BufDesc`
- `IVIDDEC2_Fxns`
- `IVIDDEC2_Params`
- `IVIDDEC2_DynamicParams`
- `IVIDDEC2_InArgs`
- `IVIDDEC2_Status`
- `IVIDDEC2_OutArgs`

4.2.1.1 XDM_BufDesc

Description

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

Fields

Field	Datatype	Input/ Output	Description
`**bufs`	`XDAS_Int8`	Input	Pointer to the vector containing buffer addresses
`numBufs`	`XDAS_Int32`	Input	Number of buffers
`*bufSizes`	`XDAS_Int32`	Input	Size of each buffer in bytes

4.2.1.2 XDM1_BufDesc

Description

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

Fields

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

4.2.1.3 XDM_SingleBufDesc

Description

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

Fields

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

4.2.1.4 XDM1_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
bufSize	XDAS_Int32	Input	Size of the buffer in bytes
accessMask	XDAS_Int32	Output	If the buffer was not accessed by the algorithm processor (For example, it was filled by DMA or other hardware accelerator that does not write through the algorithm's CPU), then no bits in this mask should be set.

4.2.1.5 XDM_AlgBufInfo

Description

This structure defines the buffer information descriptor for input and output buffers. This structure is filled when you invoke the `control()` function with the `XDM_GETBUFINFO` command.

Fields

Field	Datatype	Input/ Output	Description
minNumInBufs	XDAS_Int32	Output	Number of input buffers
minNumOutBufs	XDAS_Int32	Output	Number of output buffers
minInBufSize[XDM_MAX_IO_BUFFERS]	XDAS_Int32	Output	Size in bytes required for each input buffer
minOutBufSize[XDM_MAX_IO_BUFFERS]	XDAS_Int32	Output	Size in bytes required for each output buffer

Note:

The buffer details for MPEG2 Main Profile Decoder,are:

- Number of input buffer required is 1.
- Number of output buffer required is 2 for YUV420 interleaved.
- 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 1080p format are:

  For YUV 420 interleaved:
  Y buffer = 1920 * 1088
  UV buffer = 1920 * 544

These are the maximum buffer sizes but they can be reconfigured depending on the format of the bit-stream.

4.2.1.6 IVIDEO1_BufDesc

‖ Description

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

‖ Fields

Field	Datatype	Input/ Output	Description
numBufs	XDAS_Int32	Input	Number of buffers
frameWidth	XDAS_Int32	Input	Width of the video frame
frameHeight	XDAS_Int32	Input	Height of the video frame
framePitch	XDAS_Int32	Input	Frame pitch used to store the frame
bufDesc[IVIDEO_MAX_YUV_BUFFERS]	XDM1_SingleBufDesc	Input	Pointer to the vector containing buffer addresses
extendedError	XDAS_Int32	Input	Extended Error Field
frameType	XDAS_Int32	Input	Indicates the decoded frame type as IVIDEO_FrameType enumerator type
topFieldFirstFlag	XDAS_Int32	Input	Flag to indicate when the application should display the top field first
repeatFirstFieldFlag	XDAS_Int32	Input	Flag to indicate when the first field should be repeated
frameStatus	XDAS_Int32	Input	Frame status of IVIDEO_Output

Field	Datatype	Input/ Output	Description
repeatFrame	XDAS_Int32	Input	Number of times the display process needs to repeat the displayed progressive frame
contentType	XDAS_Int32	Input	Content type of the buffer
chromaFormat	XDAS_Int32	Input	XDM_ChromaFormat

4.2.1.7 IVIDDEC2_Fxns

Description

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

Fields

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

4.2.1.8 IVIDDEC2_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
maxWidth	XDAS_Int32	Input	Maximum video width to be supported in pixels

Field	Datatype	Input/ Output	Description
maxFrameRate	XDAS_Int32	Input	Maximum frame rate in fps * 1000 to be supported.
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.
dataEndianness	XDAS_Int32	Input	Endianness of input data. See XDM_DataFormat enumeration for details.
forceChromaFormat	XDAS_Int32	Input	Sets the output to the specified format. (Only 420 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. See XDM_ChromaFormat enumeration for details.

Note:

- MPEG2 Decoder does not use the maxFrameRate and maxBitRate fields for creating the algorithm instance.
- Maximum video height and width supported are 1088 pixels and 1920 pixels respectively (for 1080p).
- dataEndianness field should be set to XDM_BYTE.

4.2.1.9 IVIDDEC2_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
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: - 0 (XDM_DECODE_AU) - Decode entire frame including all the headers - 1 (XDM_PARSE_HEADER) - Decode only one NAL unit (Not Supported)

Field	Datatype	Input/ Output	Description
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.
frameSkipMode	XDAS_Int32	Input	Frame skip mode. See IVIDEO_FrameSkip enumeration for details.
frameOrder	XDAS_Int32	Input	Frame Display Order.
newFrameFlag	XDAS_Int32	Input	Flag to indicate that the algorithm should start a new frame. Valid values are XDAS_TRUE and XDAS_FALSE. 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.
mbDataFlag	XDAS_Int32	Input	Flag to indicate that the algorithm should generate MB Data in addition to decoding the data

Note:

- MPEG2 Decoder does not support displayWidth field.
- Frame skip is not supported. Set the frameSkipMode field to IVIDEO_NO_SKIP.
- MPEG2 Decoder always sets the output frames in the display order.
- MPEG2 Decoder does not support newFrameFlag in this version.
- MPEG2 Decoder generates MB data from a location specified in the mbDataBuf field of outargs. The MB data buffer size asked by the algorithm is currently for 14 MBs more than the total MBs in the picture. This is because the pipe-up and pipe-down stages of concurrent flow also output some redundant data.
- If mbDataFlag is 1 for an interlaced picture involving two separate fields, the MB data generated for the second field of the picture in the MB data buffer will be at an offset of 0x50 from the end of the first field's last MB data.

4.2.1.10 IVIDDEC2_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 IVIDDEC2_OutArgs data structure will be same as inputID field.

Note:

Mpeg2 Decoder copies the inputID value to the outputID value of IVIDDEC2_OutArgs structure after factoring in a display delay of 1.

4.2.1.11 IVIDDEC2_Status

Description

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

Fields

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

Field	Datatype	Input/ Output	Description
data	XDM_SingleBufDesc	Output	Buffer information structure for information passing buffer.
maxNumDisplayBufs	XDAS_Int32	Output	The maximum number of buffers required by the codec.
outputHeight	XDAS_Int32	Output	Output height in pixels
outputWidth	XDAS_Int32	Output	Output width in pixels
frameRate	XDAS_Int32	Output	Average frame rate in fps * 1000. The average frame rate for all video decoders is 30 fps.
bitRate	XDAS_Int32	Output	Average bit rate in bits per second
contentType	XDAS_Int32	Output	Video content. See IVIDEO_ContentType enumeration for details.
outputChromaFormat	XDAS_Int32	Output	Output chroma format. See XDM_ChromaFormat enumeration for details.
bufInfo	XDM_AlgBufInfo	Output	Input and output buffer information. See XDM_AlgBufInfo data structure for details.

Note:

MPEG2 Decoder does not not use the buffer descriptor meant for passing additional information between the application and the decoder.

4.2.1.12 IVIDDEC2_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.
bytesConsumed	XDAS_Int32	Output	Bytes consumed per decode call
outputID[XDM_MAX_IO	XDAS_Int32	Output	Output ID corresponding to displayBufs A value of zero (0) indicates an invalid ID. The first

Field	Datatype	Input/ Output	Description
_BUFFERS]			zero entry in array will indicate end of valid outputIDs within the array. Hence the application can stop reading the array when it encounters the first zero entry
decodedBufs	IVIDEO1_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
displayBufs[XDM_MAX_IO_BUFFERS]	IVIDEO1_BufDesc	Output	Array containing display frames corresponding to valid ID entries in the outputID array.
outputMbDataID	XDAS_Int32	Output	Output ID corresponding with the MB Data
mbDataBuf	XDM1_SingleBufDesc	Output	The decoder populates the last buffer among the buffers supplied within outBufs->bufs[] with the decoded MB data generated by the Decoder module
freeBufID[IVIDDEC2_FREE_BUFF_SIZE]	XDAS_Int32	Output	This is an array of inputIDs corresponding to the frames that have been unlocked in the 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.

The MB data of each MB is present in the output location, specified in the `mbDataBuf` field of `outArgs` structure. The format of the MB data is provided in the following table.

Byte Offset Address	Bit	Data Name
0x00	63:56	slice_vertical_position
	55:52	Reserved
	51	motion_vertical_field_select[1][1]
	50	motion_vertical_field_select[1][0]
	49	motion_vertical_field_select[0][1]
	48	motion_vertical_field_select[0][0]
	47	Reserved
	46	last_mb_in_slice
	45	skipped_mb
	44	cond_skip_flag
	43:42	motion_vector_count
	41:40	Prediction Type
	39:37	Reserved
	36:32	quantiser_scale_code
	31:18	Reserved
	17:16	dct_type
	15:5	Reserved
	4	macroblock_quant
	3	macroblock_motion_forward
	2	macroblock_motion_backward

Byte Offset Address	Bit	Data Name
	1	macroblock_pattern
	0	macroblock_intra
0x08	63:48	vector'[0][1][1]
	47:32	vector'[0][1][0]
	31:16	vector'[0][0][1]
	15:0	vector'[0][0][0]
0x10	63:48	vector'[1][1][1]
	47:32	vector'[1][1][0]
	31:16	vector'[1][0][1]
	15:0	vector'[1][0][0]
0x18	63:48	dmvector[1]
	47:32	dmvector[0]
	31:16	vector'[2][0][1]
	15:0	vector'[2][0][0]
0x20	63:48	Reserved
	47:32	Reserved
	31:16	vector'[3][0][1]
	15:0	vector'[3][0][0]

4.2.2 MPEG2 Decoder Data Structures

This section includes the following MPEG2 decoder specific data structures:

- `Imp2VDEC_Params`
- `Imp2VDEC_DynamicParams`
- `Imp2VDEC_InArgs`
- `Imp2VDEC_Status`
- `Imp2VDEC_OutArgs`

4.2.2.1 Imp2VDEC _Params

‖ Description

This structure defines the creation parameters and any other implementation specific parameters for an MPEG2 decoder instance object. The creation parameters are defined in the XDM data structure, `IVIDDEC2_Params`.

‖ Fields

Field	Datatype	Input/ Output	Description
`viddecParams`	`IVIDDEC2_Params`	Input	See `IVIDDEC2_Params` data structure for details.

4.2.2.2 Imp2VDEC_DynamicParams

‖ Description

This structure defines the run-time parameters and any other implementation specific parameters for an MPEG2 instance object. The run-time parameters are defined in the XDM data structure, `IVIDDEC2_DynamicParams`.

‖ Fields

Field	Datatype	Input/ Output	Description
`viddecDynamicParams`	`IVIDDEC2_DynamicParams`	Input	See `IVIDEC1_DynamicParams` data structure for details.

4.2.2.3 Imp2VDEC_InArgs

Description

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

Fields

Field	Datatype	Input/ Output	Description
viddecInArgs	IVIDDEC2_ _InArgs	Input	See IVIDDEC2_InArgs data structure for details.

4.2.2.4 Imp2VDEC_Status

Description

This structure defines parameters that describe the status of the MPEG2 decoder and any other implementation specific parameters. The status parameters are defined in the XDM data structure, IVIDDEC2_Status.

Fields

Field	Datatype	Input/ Output	Description
viddecStatus	IVIDDEC2_Status	Output	See IVIDDEC2_Status data structure for details

4.2.2.5 Imp2VDEC_OutArgs

Description

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

Fields

Field	Datatype	Input/ Output	Description
viddecOutArgs	IVIDDEC2_OutArgs	Output	See IVIDDEC2_OutArgs data structure for details.

4.3 Interface Functions

This section describes the application programming interfaces used in the MPEG2 decoder. The MPEG2 decoder 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.

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.

‖ 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 func-
tions */

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. Since the client 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.

‖ See Also

`algNumAlloc(), algFree()`

4.3.2 Initialization API

Initialization API is used to initialize an instance of the MPEG2 decoder. The initialization parameters are defined in the `IVIDDEC2_Params` structure (see Data Structures section for details).

‖ Name

`algInit()` – initialize an algorithm instance

‖ Synopsis

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

‖ Arguments

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

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

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

IALG_Params *params; /* algorithm initialization parame-
ters */
```

‖ 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. All fields in the `params` structure must be set as described in `IALG_Params` structure (see Data Structures section for details).

For more details, see TMS320 DSP Algorithm Standard API Reference.

‖ See Also

```
algAlloc(), algMoved()
```

4.3.3 Control API

Control API is used for controlling the functioning of MPEG2 decoder 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 `IVIDDEC2_Status` data structure (see Data Structures section for details).

‖ Name

`control()` – change run-time parameters of the MPEG2 decoder and query the decoder status

‖ Synopsis

```
XDAS_Int32 (*control)(IVIDDEC2_Handle handle, IVIDDEC2_Cmd
id,IVIDDEC2_DynamicParams *params, IVIDDEC2_Status
*status);
```

‖ Arguments

```
IVIDDEC2_Handle handle; /* handle to the MPEG2 decoder
instance */

IVIDDEC2_Cmd id; /* MPEG2 decoder specific control
commands*/

IVIDDEC2_DynamicParams *params /* MPEG2 decoder run-time
parameters */

IVIDDEC2_Status *status /* MPEG2 decoder instance status
parameters */
```

‖ Return Value

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

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

‖ Description

This function changes the run-time parameters of MPEG2 decoder and queries the status of decoder. `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 the MPEG2 decoder instance object.

The second argument is a command ID. See `XMI_CmdId` in enumeration table for details.

The third and fourth arguments are pointers to the `IVIDDEC2_DynamicParams` and `IVIDDEC2_Status` data structures respectively.

‖ See Also

`algInit()`

4.3.4 Data Processing API

Data processing API is used for processing the input data using the MPEG2 decoder.

‖ 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 video decoding call

‖ Synopsis

```
XDAS_Int32 (*process)(IVIDDEC2_Handle handle, XDM1_BufDesc
*inBufs, XDM_BufDesc *outBufs, IVIDDEC2_InArgs *inargs,
IVIDDEC2_OutArgs *outargs);
```

‖ Arguments

```
IVIDDEC2_Handle handle; /* handle to the MPEG2 decoder
instance */

XDM1_BufDesc *inBufs; /* pointer to input buffer descrip-
tor data structure */

XDM_BufDesc *outBufs; /* pointer to output buffer descrip-
tor data structure */

IVIDDEC2_InArgs *inargs /* pointer to the MPEG2 decoder
runtime input arguments data structure */

IVIDDEC2_OutArgs *outargs /* pointer to the MPEG2 decoder
runtime output arguments data structure */
```

‖ Return Value

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

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

‖ Description

This function does the basic MPEG2 video decoding. The first argument to `process()` is a handle to the MPEG2 decoder instance object.

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 `IVIDDEC2_InArgs` data structure that defines the runtime input arguments for the MPEG2 decoder instance object.

> **Note**:
>
> Prior to each decode call, ensure that all fields are set as described in `XDM_BufDesc` and `IVIDDEC2_InArgs` structures.

The last argument is a pointer to the `IVIDDEC2_OutArgs` data structure that defines the runtime output arguments for the MPEG2 decoder instance object.

The algorithm may also modify the output buffer pointers. The return value is `IALG_EOK` for success or `IALG_EFAIL` in case of failure. The `extendedError` field of the `IVIDDEC2_OutArgs` structure contains further error conditions flagged by the algorithm.

‖ See Also

`control()`

‖ 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.

‖ See Also

`algActivate()`

4.3.5 Termination API

Termination API is used to terminate the MPEG2 decoder 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 mem-
Tab[]);
```

‖ Arguments

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

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

‖ Return Value

```
XDAS_Int32; /* Number of buffers used by the algorithm */
```

‖ Description

`algFree()` determines the addresses of all memory buffers used by the algorithm. The primary aim of doing so is to free up these memory regions after closing an instance of the algorithm.

The first argument to `algFree()` is a handle to the algorithm instance.

The second argument is a table of memory records that describe the base address, size, alignment, type, and memory space of all buffers previously allocated for the algorithm instance.

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

‖ See Also

`algAlloc()`