

MPEG2 Main Profile Decoder on C64x+

User's Guide



Literature Number: SPRUEA4D
April 2010

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

Products

| | |
|-----------------------------|--|
| Amplifiers | amplifier.ti.com |
| Data Converters | dataconverter.ti.com |
| DLP® Products | www.dlp.com |
| DSP | dsp.ti.com |
| Clocks and Timers | www.ti.com/clocks |
| Interface | interface.ti.com |
| Logic | logic.ti.com |
| Power Mgmt | power.ti.com |
| Microcontrollers | microcontroller.ti.com |
| RFID | www.ti-rfid.com |
| RF/IF and ZigBee® Solutions | www.ti.com/lprf |
| Wireless | www.ti.com/wireless-apps |

Applications

| | |
|----------------------------|--|
| Audio | www.ti.com/audio |
| Automotive | www.ti.com/automotive |
| Communications and Telecom | www.ti.com/communications |
| Computers and Peripherals | www.ti.com/computers |
| Consumer Electronics | www.ti.com/consumer-apps |
| Energy | www.ti.com/energy |
| Industrial | www.ti.com/industrial |
| Medical | www.ti.com/medical |
| Security | www.ti.com/security |
| Space, Avionics & Defense | www.ti.com/space-avionics-defense |
| Video and Imaging | www.ti.com/video |

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2010, Texas Instruments Incorporated

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 C64x+ 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 C64x+ platform.

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

How to Use This Manual

This document includes the following chapters:

- ❑ **Chapter 1 - Introduction**, provides a brief introduction to the XDAIS and XDM standards. It also provides an overview of the codec and lists its supported features.
- ❑ **Chapter 2 - Installation Overview**, describes how to install, build, and run the codec.
- ❑ **Chapter 3 - Sample Usage**, describes the sample usage of the codec.
- ❑ **Chapter 4 - API Reference**, describes the data structures and interface functions used in the codec.
- ❑ **Chapter 5 - Frequently Asked Questions**, provides answers to few frequently asked questions related to using this decoder.
- ❑ **Appendix A – Revision History**, highlights the changes made to the SPRUEA4C codec specific user guide to make it SPRUEA4D

Related Documentation From Texas Instruments

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

- ❑ *TMS320 DSP Algorithm Standard Rules and Guidelines* (literature number SPRU352) defines a set of requirements for DSP algorithms that, if followed, allow system integrators to quickly assemble production-quality systems from one or more such algorithms.
- ❑ *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360) describes all the APIs that are defined by the TMS320 DSP Algorithm Interface Standard (also known as XDAIS) specification.
- ❑ *Technical Overview of eXpressDSP - Compliant Algorithms for DSP Software Producers* (literature number SPRA579) describes how to make algorithms compliant with the TMS320 DSP Algorithm Standard which is part of TI's eXpressDSP technology initiative.
- ❑ *Using the TMS320 DSP Algorithm Standard in a Static DSP System* (literature number SPRA577) describes how an eXpressDSP-compliant algorithm may be used effectively in a static system with limited memory.
- ❑ *DMA Guide for eXpressDSP-Compliant Algorithm Producers and Consumers* (literature number SPRA445) describes the DMA architecture specified by the TMS320 DSP Algorithm Standard (XDAIS). It also describes two sets of APIs used for accessing DMA resources: the IDMA2 abstract interface and the ACPY2 library.
- ❑ *eXpressDSP Digital Media (XDM) Standard API Reference* (literature number SPRUEC8)

The following documents describe TMS320 devices and related support tools:

- ❑ *Design and Implementation of an eXpressDSP-Compliant DMA Manager for C6X1X* (literature number SPRA789) describes a C6x1x-optimized (C6211, C6711) ACPY2 library implementation and DMA Resource Manager.
- ❑ *TMS320C64x+ Megamodule* (literature number SPRAA68) describes the enhancements made to the internal memory and describes the new features which have been added to support the internal memory architecture's performance and protection.
- ❑ *TMS320C64x+ DSP Megamodule Reference Guide* (literature number SPRU871) describes the C64x+ megamodule peripherals.
- ❑ *TMS320C64x to TMS320C64x+ CPU Migration Guide* (literature number SPRAA84) describes migration from the Texas Instruments TMS320C64x™ digital signal processor (DSP) to the TMS320C64x+™ DSP.

- ❑ *TMS320C6000 Optimizing Compiler v 6.0 Beta User's Guide* (literature number SPRU187N) explains how to use compiler tools such as compiler, assembly optimizer, standalone simulator, library-build utility, and C++ name demangler.
- ❑ *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (literature number SPRU732) describes the CPU architecture, pipeline, instruction set, and interrupts of the C64x and C64x+ DSPs.
- ❑ *TMS320DM6446 Digital Media System-on-Chip* (literature number SPRS283)
- ❑ *TMS320DM6446 Digital Media System-on-Chip Errata (Silicon Revision 1.0)* (literature number SPRZ241) describes the known exceptions to the functional specifications for the TMS320DM6446 Digital Media System-on-Chip (DMSoC).
- ❑ *TMS320DM6443 Digital Media System-on-Chip* (literature number SPRS282)
- ❑ *TMS320DM6443 Digital Media System-on-Chip Errata (Silicon Revision 1.0)* (literature number SPRZ240) describes the known exceptions to the functional specifications for the TMS320DM6443 Digital Media System-on-Chip (DMSoC).
- ❑ *TMS320DM644x DMSoC DSP Subsystem Reference Guide* (literature number SPRUE15) describes the digital signal processor (DSP) subsystem in the TMS320DM644x Digital Media System-on-Chip (DMSoC).
- ❑ *TMS320DM644x DMSoC ARM Subsystem Reference Guide* (literature number SPRUE14) describes the ARM subsystem in the TMS320DM644x Digital Media System on a Chip (DMSoC).
- ❑ *MPEG-2 Video Decoder: TMS320C62x DSP Implementation* (literature number SPRA649) describes the implementation of the MPEG-2 video decoder on the TMS320C62x DSP.

Related Documentation

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

- ❑ *ISO/IEC 11172-2 Information technology -- Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbits/s -- Part 2: Video (MPEG-1 video standard).*
- ❑ *ISO/IEC 13818-2 Information technology -- Generic coding of moving pictures and associated audio information: Video (MPEG-2 video standard).*

Abbreviations

The following abbreviations are used in this document.

Table 1-1. List of Abbreviations

| Abbreviation | Description |
|--------------|---|
| API | Application Programming Interface |
| CBR | Constant Bit Rate |
| CPB | Constrained Parameters Bit-streams |
| DCT | Discrete Cosine Transform |
| DMA | Direct Memory Access |
| DMAN3 | DMA Manager |
| DSS | Direct Satellite System |
| DTV | Digital Television |
| DVB | Digital Video Broadcast |
| DVD | Digital Versatile Disc |
| EVM | Evaluation Module |
| fps | Frames per second |
| IDCT | Inverse Discrete Cosine Transform |
| Kbps | Kilo bits per second |
| MPEG | Motion Picture Expert Group |
| VCL | Video Coding Layer |
| XDAIS | eXpressDSP Algorithm Interface Standard |
| XDM | eXpressDSP Digital Media |

Text Conventions

The following conventions are used in this document:

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

Product Support

When contacting TI for support on this codec, quote the product name (MPEG2 Main Profile Decoder on C64x+) and version number. The version number of the codec is included in the title of the release notes that accompanies this codec.

Trademarks

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

All trademarks are the property of their respective owners.

This page is intentionally left blank

Contents

| | |
|---|-------------|
| Read This First | iii |
| About This Manual | iii |
| Intended Audience | iii |
| How to Use This Manual | iii |
| Related Documentation From Texas Instruments..... | iv |
| Related Documentation..... | v |
| Abbreviations | vi |
| Text Conventions | vi |
| Product Support | vii |
| Trademarks | vii |
| Contents..... | ix |
| Figures | xi |
| Tables..... | xiii |
| Introduction | 1-1 |
| 1.1 Overview of XDAIS and XDM..... | 1-2 |
| 1.1.1 XDAIS Overview | 1-2 |
| 1.1.2 XDM Overview | 1-2 |
| 1.2 Overview of MPEG2 Main Profile Decoder | 1-3 |
| 1.3 Supported Services and Features..... | 1-4 |
| Installation Overview | 2-1 |
| 2.1 System Requirements | 2-2 |
| 2.1.1 Hardware..... | 2-2 |
| 2.1.2 Software | 2-2 |
| 2.2 Installing the Component..... | 2-2 |
| 2.3 Before Building the Sample Test Application | 2-4 |
| 2.3.1 Installing DSP/BIOS | 2-4 |
| 2.3.2 Installing Framework Component (FC) | 2-4 |
| 2.3.3 Installing Codec Engine (CE) | 2-5 |
| 2.4 Building and Running the Sample Test application..... | 2-5 |
| 2.5 Configuration Files | 2-6 |
| 2.5.1 Generic Configuration File | 2-6 |
| 2.5.2 Decoder Configuration File | 2-7 |
| 2.6 Standards Conformance and User-Defined Inputs | 2-8 |
| 2.7 Uninstalling the Component | 2-8 |
| 2.8 Evaluation Version | 2-8 |
| Sample Usage..... | 3-1 |
| 3.1 Overview of the Test Application..... | 3-2 |
| 3.1.1 Parameter Setup | 3-3 |
| 3.1.2 Algorithm Instance Creation and Initialization..... | 3-3 |
| 3.1.3 Process Call | 3-4 |
| 3.1.4 Algorithm Instance Deletion | 3-5 |
| 3.2 Frame Buffer Management by Application | 3-6 |
| 3.2.1 Frame Buffer Input and Output | 3-6 |
| 3.2.2 Frame Buffer Management by Application..... | 3-8 |

| | |
|---|------------|
| 3.3 Sample Test Application..... | 3-9 |
| API Reference..... | 4-1 |
| 4.1 Symbolic Constants and Enumerated Data Types..... | 4-2 |
| 4.2 Behavioral Specification of the Decoder | 4-7 |
| 4.2.1 Classification of Errors | 4-7 |
| 4.2.2 Recommended Steps to Recover from Error..... | 4-10 |
| 4.3 Data Structures | 4-12 |
| 4.3.1 Common XDM Data Structures..... | 4-12 |
| 4.3.2 MPEG2 Decoder Data Structures | 4-23 |
| 4.4 Interface Functions..... | 4-31 |
| 4.4.1 Creation APIs | 4-31 |
| 4.4.2 Initialization API..... | 4-33 |
| 4.4.3 Control API..... | 4-34 |
| 4.4.4 Data Processing API..... | 4-36 |
| 4.4.5 Termination API | 4-40 |
| Frequently Asked Questions | 5-1 |

Figures

| | |
|---|-----|
| Figure 2-1. Component Directory Structure | 2-2 |
| Figure 3-1. Test Application Sample Implementation..... | 3-2 |
| Figure 3-3. Frame Buffer Pointer Implementation..... | 3-7 |
| Figure 3-2. Interaction of Frame Buffers Between Application and Framework..... | 3-8 |

This page is intentionally left blank

Tables

| |
|--|
| |
| |
| |

| | |
|---|-----|
| Table 1-1. List of Abbreviations..... | vi |
| Table 2-1. Component Directories..... | 2-3 |
| Table 4-1. List of Enumerated Data Types..... | 4-2 |
| Table 4-2. MPEG2 Decoder Fatal Error Statuses | 4-7 |
| Table 4-3. MPEG2 Decoder Non-Fatal Bit stream Error Statuses..... | 4-7 |
| Table 4-4. MPEG2 Decoder Non-Fatal Error Statuses | 4-9 |
| Table 5-1. FAQs for MPEG2 Decoder on C64x+ | 5-1 |
| Table A-1. Revision History of MPEG2 Decoder on C64x+ | 5-1 |

This page is intentionally left blank

Introduction

This chapter provides a brief introduction to XDAIS and XDM. It also provides an overview of TI's implementation of the MPEG2 Main Profile Decoder on the C64x+ platform and its supported features.

| Topic | Page |
|--|------|
| 1.1 Overview of XDAIS and XDM | 1-2 |
| 1.2 Overview of MPEG2 Main Profile Decoder | 1-3 |
| 1.3 Supported Services and Features | 1-4 |

1.1 Overview of XDAIS and XDM

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

1.1.1 XDAIS Overview

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

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

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

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

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

1.1.2 XDM Overview

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

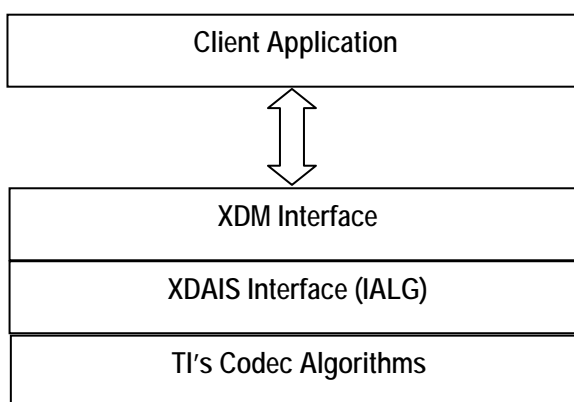
(For example audio, video, image, and speech). The XDM standard defines the following two APIs:

- ❑ `control()`
- ❑ `process()`

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

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

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



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

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

1.2 Overview of MPEG2 Main Profile Decoder

The MPEG2 video standard specifies 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.

The decoder software implements all the MPEG2 main-profile-at-high-level functionality. For more information on the MPEG2 video decoding algorithm, see *MPEG2 Video Decoder: TMS320C62x (TM) DSP Implementation* application report (literature number SPRA649).

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 Decoder on the C64x+ platform. This version of the codec has the following supported features:

- ❑ Supports MPEG2 Main Profile at high level feature of the ISO/IEC 13818-2 standard
- ❑ Compliant as per ISO/IEC 13818-4 conformance standard, based on Inverse Discrete Cosine Transform (IDCT)
- ❑ Supports YUV 420 planar and YUV 422 interleaved output formats
- ❑ Supports interlace and progressive decoding
- ❑ Supports only elementary video stream input formats
- ❑ Supports MPEG-1 Constrained Parameters Bit-streams (CPB)
- ❑ Supports bottom field reordering in case of non-progressive sequences, where bottom field is sent ahead of top field for frame pictures
- ❑ Supports trick play and reverse play
- ❑ Supports display width feature
- ❑ Supports streams that are non-multiples of 16
- ❑ Supports feature `XDM_PARSE_HEADER`. It allows parsing of only the headers, skipping the picture data decoding
- ❑ eXpressDSP Digital Media (XDM 1.0 IVIDDEC2) compliant

Note:

This version of MPEG2 Decoder does not support sequences with escape bit set in the sequence extension header.

Installation Overview

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

| Topic | Page |
|--|------|
| 2.1 System Requirements | 2-2 |
| 2.2 Installing the Component | 2-2 |
| 2.3 Before Building the Sample Test Application | 2-4 |
| 2.4 Building and Running the Sample Test application | 2-5 |
| 2.5 Configuration Files | 2-6 |
| 2.6 Standards Conformance and User-Defined Inputs | 2-8 |
| 2.7 Uninstalling the Component | 2-8 |
| 2.8 Evaluation Version | 2-8 |

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 DM644x EVM with XDS560 JTAG emulator.

The codec library will run as is on the following devices : DM6446, DM6437, DM6467, OMAP3530. (Note: L1-D, L1-P and DDR addresses may be different across the devices).

2.1.2 Software

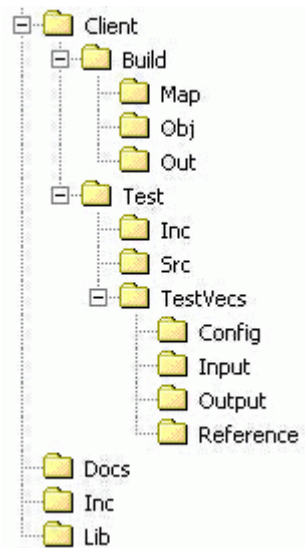
The following are the software requirements for the normal functioning of the codec:

- ❑ **Development Environment:** This project is developed using Code Composer Studio version 3.3.49.
- ❑ **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 100_V_MPEG2_D_02_00, under which another directory named C64XPLUS_MP_001 is created.

Figure 2-1 shows the sub-directories created in the C64XPLUS_MP_001 directory.



Note: If you are installing an evaluation version of this codec, the directory name will be 100E_V_MPEG2_D_02_00.

Table 2-1 provides a description of the sub-directories created in the C64XPLUS_MP_001 directory.

Table 2-1. Component Directories

| Sub-Directory | Description |
|---------------------------------|--|
| \Inc | Contains XDM related header files which allow interface to the codec library |
| \Lib | Contains the codec library file |
| \Docs | Contains user guide and datasheet |
| \Client\Build | Contains the sample test application project (.pj1) file |
| \Client\Build\Map | Contains the memory map generated on compilation of the code |
| \Client\Build\Obj | Contains the intermediate .asm and/or .obj file generated on compilation of the code |
| \Client\Build\Out | Contains the final application executable (.out) file generated by the sample test application |
| \Client\Test\Src | Contains application C files |
| \Client\Test\Inc | Contains header files needed for the application code |
| \Client\Test\TestVecs\Input | Contains input test vectors |
| \Client\Test\TestVecs\Output | Contains output generated by the codec |
| \Client\Test\TestVecs\Reference | Contains read-only reference output to be used for verifying against codec output |
| \Client\Test\TestVecs\Config | Contains configuration parameter files |

2.3 Before Building the Sample Test Application

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

This version of the codec has been validated with DSP/BIOS version 5.32.02, Codec Engine 2.00.01, and Framework Component (FC) version 2.20.00.15.

2.3.1 Installing DSP/BIOS

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

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

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

<install directory>\CCStudio_v3.3

The sample test application uses 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 Framework Component (FC)

You can download FC from the TI external website:

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

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

<install directory>\CCStudio_v3.3

The test application uses the following DMAN3 files:

- ❑ Library file, dman3.a64P available in the
<install directory>\CCStudio_v3.3<fc_directory>\packages
 \ti\sdo\fc\dman3 directory.
- ❑ Header file, dman3.h available in the
<install directory>\CCStudio_v3.3<fc_directory>\packages
 \ti\sdo\fc\dman3 directory.
- ❑ Header file, idma3.h available in the
<install directory>\CCStudio_v3.3<fc_directory>\fctools\packages
 \ti\xdais directory.

2.3.3 Installing Codec Engine (CE)

Download CE version 2.00.01 from TI's external website:

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

The codec uses framework components and XDAIS version that are a part of CE 2.00.01.

To install CE:

- 1) Extract the CE zip file to the same location where you have installed Code Composer Studio. For example:

<install direc-tory>\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.
 - 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.

2.4 Building and Running the Sample Test application

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

- 1) Verify that you have installed TI's Code Composer Studio version 3.3.49 and code generation tools version 6.0.8.
- 2) Verify that the codec object library, `mpeg2vdec_ti.l64P` exists in the \Lib sub-directory.
- 3) Open the test application project file, `TestAppDecoder.pjt` in Code Composer Studio. This file is available in the \Client\Build sub-directory.

- 4) Select **Project > Build** to build the sample test application. This creates an executable file, TestAppDecoder.out in the \Client\Build\Out sub-directory.
- 5) Select **File > Load**, browse to the \Client\Build\Out sub-directory, select the codec executable created in step 4, and load it into Code Composer Studio in preparation for execution.
- 6) Select **Debug > Run** to execute the sample test application.
The sample test application takes the input files stored in the \Client\Test\TestVecs\Input sub-directory, runs the codec, and uses the reference files stored in the \Client\Test\TestVecs\Reference sub-directory to verify that the codec is functioning as expected.
- 7) On successful completion, the application displays one of the following messages for each frame:
 - “Decoder compliance test passed/failed”(for compliance check mode)
Pass criteria is that, more than 99 percent decoded samples should have at the most one-bit difference with reference output satisfied.
 - “Decoder output dump completed” (for output dump mode)

2.5 Configuration Files

This codec is shipped along with:

- ❑ Generic configuration file (Testvecs.cfg) – specifies input and reference files for the sample test application.
- ❑ Decoder configuration file (Testparams.cfg) – specifies the configuration parameters used by the test application to configure 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
 - 0 - for writing the output to the output file
- ❑ `Config` is the Decoder configuration file. For details, see Section 2.5.2.

- ❑ Input is the input file name (use complete path).
- ❑ Output/Reference is the output file name (if x is 0) or reference file name (if X is 1).

A sample Testvecs.cfg file is as shown:

```
1
..\..\Test\TestVecs\Config\Testparams.cfg
..\..\Test\TestVecs\Input\davincieffect_qcif_440kbps.m2v
..\..\Test\TestVecs\Reference\davincieffect_qcif_440kbps_420Ref.yuv
0
..\..\Test\TestVecs\Config\Testparams.cfg
..\..\Test\TestVecs\Input\davincieffect_qcif_440kbps.m2v
..\..\Test\TestVecs\Output\davincieffect_qcif_440kbps.yuv
```

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:

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

ImageWidth = 1920    # Image width in Pels, must be
                    # multiples of 16
ImageHeight = 1088   # Image height in Pels, must be
                    # multiples of 16
ChromaFormat = 1     # 1 => XDM_YUV_420P,
                    # 3 => XDM_YUV_422IBE,
                    # 4 => XDM_YUV_422ILE
FramesToDecode = 10  # Number of frames to be coded
```

Any field in the `IVIDDEC2_Params` structure (see Section 4.3.1.8) 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.

2.6 Standards Conformance and User-Defined Inputs

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

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

- 1) Copy the input files to the \Client\Test\TestVecs\Inputs sub-directory.
- 2) Copy the reference files to the \Client\Test\TestVecs\Reference sub-directory.
- 3) Edit the configuration file, Testvecs.cfg available in the \Client\Test\TestVecs\Config sub-directory. For details on the format of the Testvecs.cfg file, see Section 2.5.1.
- 4) Execute the sample test application. On successful completion, the application displays one of the following message for each frame:
 - “Decoder compliance test passed/failed”(if x is 1)
Pass criteria is that, more than 99 percent decoded samples should have at the most one-bit difference with reference output satisfied.
 - “Decoder output dump completed” (if x is 0)

If you have chosen the option 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.

2.7 Uninstalling the Component

Note:

Compliance test succeeds only for the example input file provided with evaluation package due to the presence of Texas instruments logo in the output reference file. Compliance should not be checked for other inputs.

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 | 3-2 |
| 3.2 Frame Buffer Management by Application | 3-6 |
| 3.3 Sample Test Application | 3-9 |

3.1 Overview of the Test Application

The test application exercises the `IVIDDEC2` base class of the MPEG2 Decoder library. The main test application files are `TestAppDecoder.c` and `TestAppDecoder.h`. These files are available in the `\Client\Test\Src` and `\Client\Test\Inc` sub-directories respectively.

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

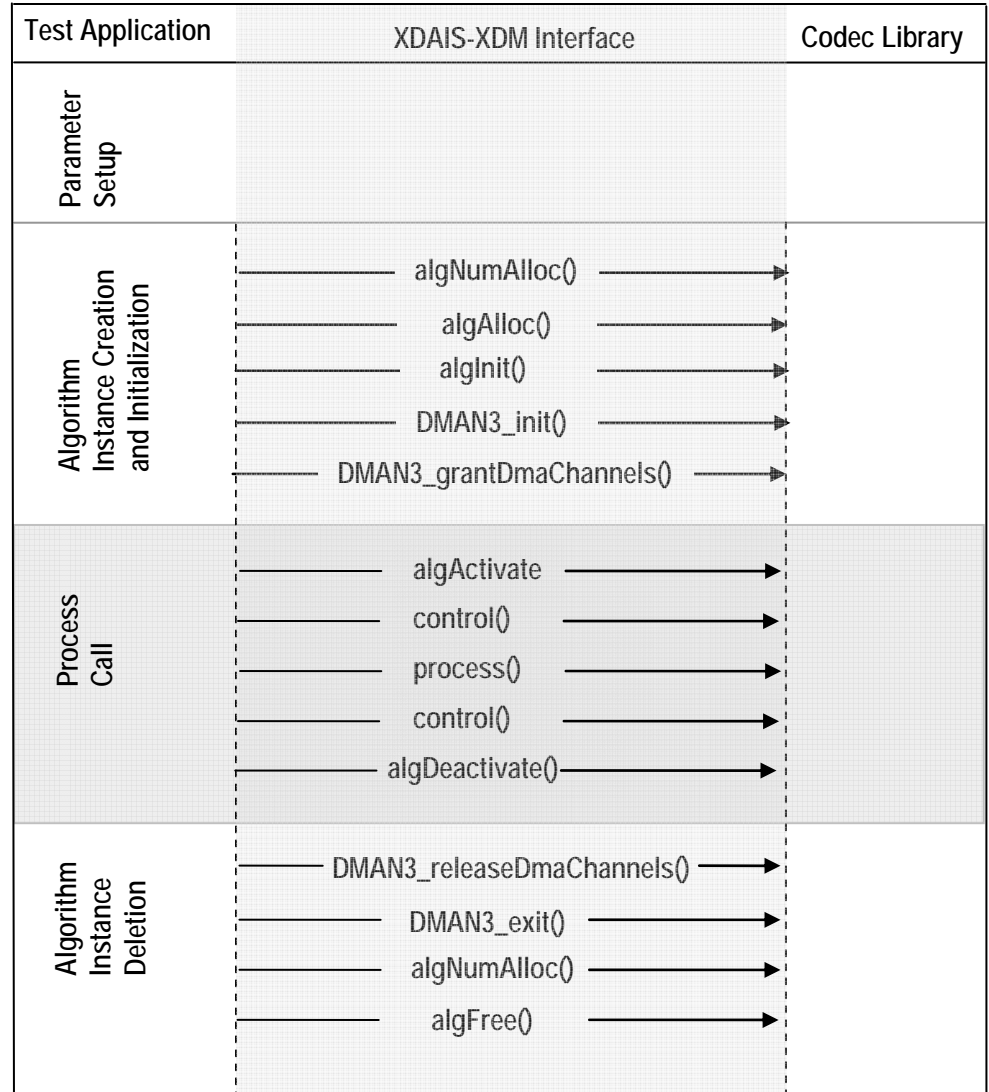


Figure 3-1. Test Application Sample Implementation

The test application is divided into four logical blocks:

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

3.1.1 Parameter Setup

Each codec component requires various codec configuration parameters to be set at initialization. For example, a video codec requires parameters such as video height, video width, and so on. The test application obtains the required parameters from the 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) Initializes the various DMAN3 parameters.
- 5) Reads the input bit-stream into the application input buffer.

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

3.1.2 Algorithm Instance Creation and Initialization

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

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

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

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

- 1) `DMAN3_init()` - To initialize the DMAN module.
- 2) `DMAN3_grantDmaChannels()` - To grant the DMA resources to the algorithm instance.

Note:

DMAN3 function implementations are provided in `dman3.a64P` library.

3.1.3 Process Call

After algorithm instance creation and initialization, the test application 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) Calls the `process()` function to encode/decode a single frame of data. The behavior of the algorithm can be controlled using various dynamic parameters (see Section 4.3.1.9). The inputs to the process function are input and output buffer descriptors, pointer to the `IVIDDEC2_InArgs` and `IVIDDEC2_OutArgs` structures.

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

- 1) `algActivate()` - To activate the algorithm instance.
- 2) `control()` (optional) - To query the algorithm on status or setting of dynamic parameters 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.

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 encoding/decoding is complete, the test application releases the DMA channels granted by the DMA Manager interface and deletes the current algorithm instance. The following APIs are called in sequence:

- 1) `DMAN3_releaseDmaChannels()` - To remove logical channel resources from an algorithm instance.
- 2) `DMAN3_exit()` - To free DMAN3 memory resources.
- 3) `algNumAlloc()` - To query the algorithm about the number of memory records it used.
- 4) `algFree()` - To query the algorithm to get the memory record information.

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

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. 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 returned 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 first frame, if the above control call returns a smaller size than the one returned in the first call.

The frame pointer given by application and that returned by 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 following figure explains the frame pointer usage.

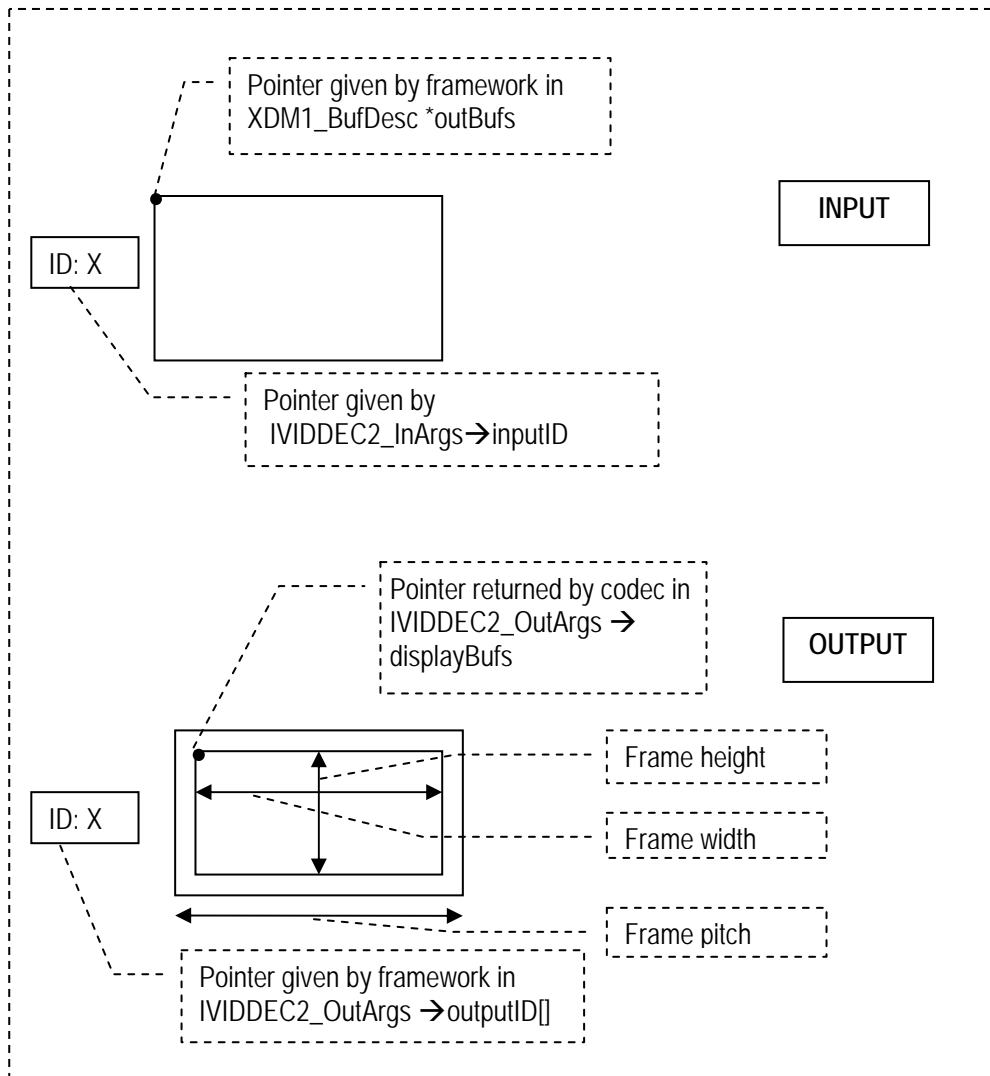


Figure 3-3. Frame Buffer Pointer Implementation

Note:

- ❑ Frame pointer returned by codec in `displayBufs` will point to actual start location of picture
- ❑ Frame height and width will be 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 unique identifier to keep record of frame buffers. Any buffer given to algorithm should be considered locked by algorithm unless the buffer is returned back to application through `IVIDDEC2_OutArgs->freeBufID[]`.

Note:

BufferID given back in `IVIDDEC2_OutArgs ->outputID[]` are just for display purpose. Application should not consider it free unless it comes as part of `IVIDDEC2_OutArgs->freeBufID[]`. In case of erroneous frame, the `freeBufID[]` returns the `inputID` given for that particular `process()` call.

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.

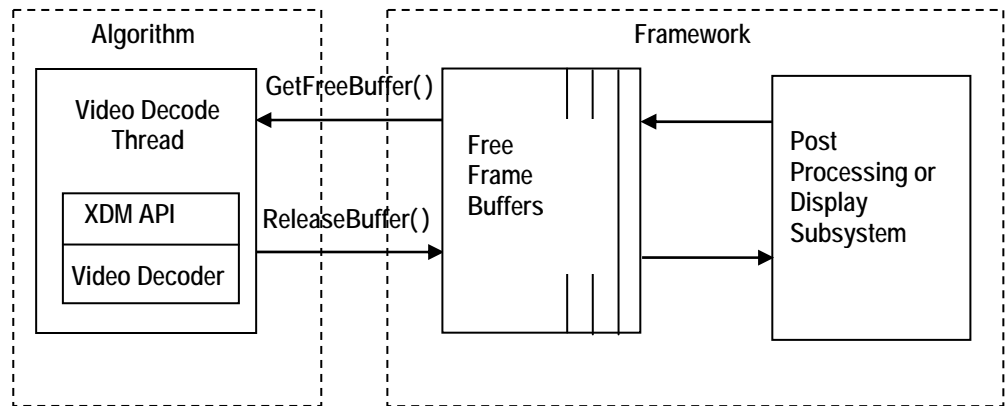


Figure 3-2. Interaction of Frame Buffers Between Application and Framework.

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()` - This 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()` - This 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()` - This function searches for a free buffer in global buffer array and returns the address of that element. If none of the elements are free, then it returns NULL.

- ❑ `BUFFMGR_ReleaseBuffer()` - This function takes an array of buffer-IDs that are released by the test-app. 0 is not a valid buffer ID. Hence, this function keeps moving until it finds a buffer ID as zero or it hits the `MAX_BUFF_ELEMENTS`.
- ❑ `BUFFMGR_DeInit()` - This function releases all memory allocated by buffer manager.

3.3 Sample Test Application

The test application exercises the `IVIDDEC2` base class of the MPEG2 Decoder. The `process()` implementation is as shown.

```
/* Main Function acting as a client for Video Decode
Call*/
BUFFMGR_Init();

TestApp_SetInitParams(&params.viddecParams);

/*----- 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
control() */
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.

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 | 4-2 |
| 4.2 Behavioral Specification of the Decoder | 4-7 |
| 4.3 Data Structures | 4-12 |
| 4.4 Interface Functions | 4-31 |

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_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 an 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. |
| | IVIDEO_FRAMETYPE_DEFAULT | Default value is set to IVIDEO_I_FRAME |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|----------------------------|-------------------------------|--|
| IVIDEO_ContentType | IVIDEO_CONTENTTYPE_NA | -1, Frame type is not available |
| | IVIDEO_PROGRESSIVE | 0, Progressive frame |
| | IVIDEO_PROGRESSIVE_FRAME | Is equal to IVIDEO_PROGRESSIVE |
| | IVIDEO_INTERLACED | 1, Interlaced frame |
| | IVIDEO_INTERLACED_FRAME | IVIDEO_INTERLACED |
| | IVIDEO_INTERLACED_TOPFIELD | 2, Interlaced picture, top field |
| | IVIDEO_INTERLACED_BOTTOMFIELD | 3, Interlaced picture, bottom field |
| IVIDEO_FrameSkip | IVIDEO_NO_SKIP | Do not skip the current frame |
| | 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 MPEG2 Decoder. |
| | 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) Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SKIP_IB | Skip I and B frame/field(s) Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SKIP_PB | Skip P and B frame/field(s) Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SKIP_IPB | Skip I/P/B/BI frames Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SKIP_IDR | Skip IDR Frame Not supported in this version of MPEG2 Decoder. |
| | IVIDEO_SKIP_DEFAULT | Default value is set to IVIDEO_NO_SKIP |
| ePicStrFormat | MPEG2VDEC_TOP_FIELD | Indicates top field in field picture |

| Group or Enumeration Class | Symbolic Constant Name | Description or Evaluation |
|----------------------------|--------------------------|--|
| XDM_DataFormat | MPEG2VDEC_BOTTOM_FIELD | Indicates bottom field in field picture |
| | MPEG2VDEC_FRAME_PICTURE | Indicates frame picture |
| | XDM_BYTE | Big endian stream |
| | XDM_LE_16 | 16-bit little endian stream. Not supported in this version of MPEG2 Decoder. |
| XDM_ChromaFormat | XDM_LE_32 | 32-bit little endian stream |
| | XDM_CHROMA_NA | -1, Chroma format not applicable |
| | XDM_YUV_420P | YUV 4:2:0 planar |
| | XDM_YUV_422P | YUV 4:2:2 planar. Not supported in this version of MPEG2 Decoder. |
| | XDM_YUV_422IBE | YUV 4:2:2 interleaved (big endian). Not supported in this version of MPEG2 Decoder. |
| | XDM_YUV_422ILE | YUV 4:2:2 interleaved (little endian) |
| | XDM_YUV_444P | YUV 4:4:4 planar. Not supported in this version of MPEG2 Decoder. |
| | XDM_YUV_411P | YUV 4:1:1 planar. Not supported in this version of MPEG2 Decoder. |
| | XDM_GRAY | Gray format. Not supported in this version of MPEG2 Decoder. |
| | XDM_RGB | RGB color format. Not supported in this version of MPEG2 Decoder. |
| XDM_CmdId | XDM_CHROMAFORMAT_DEFAULT | Default value is set to XDM_YUV_422ILE |
| | 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 |

| 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. A <code>process()</code> call has to be made after a flush call to output the stored data. |
| | XDM_GETBUFINFO | Query algorithm instance regarding the properties of input and output buffers |
| | XDM_GETVERSION | Query the algorithm's version. The result will be returned in the <code>@c</code> data field of the respective <code>_Status</code> structure. |
| XDM_DecMode | XDM_DECODE_AU | Decode entire access unit |
| | XDM_PARSE_HEADER | Decode only header |
| XDM_ErrorBit | XDM_APPLIEDCONCEALMENT | Bit 9 <input type="checkbox"/> 1 - Applied concealment <input type="checkbox"/> 0 - Ignore |
| | XDM_INSUFFICIENTDATA | Bit 10 <input type="checkbox"/> 1 - Insufficient data <input type="checkbox"/> 0 - Ignore |
| | XDM_CORRUPTEDDATA | Bit 11 <input type="checkbox"/> 1 - Data problem/corruption <input type="checkbox"/> 0 - Ignore |
| | XDM_CORRUPTEDHEADER | Bit 12 <input type="checkbox"/> 1 - Header problem/corruption <input type="checkbox"/> 0 - Ignore |
| | XDM_UNSUPPORTEDINPUT | Bit 13 <input type="checkbox"/> 1 - Unsupported feature/parameter in input <input type="checkbox"/> 0 - Ignore |
| | XDM_UNSUPPORTEDPARAM | Bit 14 <input type="checkbox"/> 1 - Unsupported input parameter or configuration <input type="checkbox"/> 0 - Ignore |
| | XDM_FATALERROR | Bit 15 <input type="checkbox"/> 1 - Fatal error (stop encoding) <input type="checkbox"/> 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 (see Section 4.2.1)

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

The MPEG2 Decoder specific error status messages are listed in Section 4.2.1. The value column indicates the decimal value of the last 8-bits reserved for codec specific error statuses.

4.2 Behavioral Specification of the Decoder

Behavioral specification defines how the decoder reacts to errors, and recommended steps to be taken by the application to recover from such errors.

4.2.1 Classification of Errors

Errors that the decoder returns can be grouped into the following categories:

- ❑ **Fatal errors** - These are non-recoverable errors that cause the application to re-initialize, reset, or re-instantiate the decoder for resuming normal operation. The application cannot continue with the decoding until the decoder is reset.
- ❑ **Non-Fatal bit stream errors** - These are errors for which the application can proceed with the decoding without resetting the decoder. The current frame is erroneous and hence decoder can continue with the decoding skipping the erroneous frame. The `process()` function returns `FAILURE`.
- ❑ **Non-Fatal errors** – These are errors for which the decoder can continue decoding the current frame. The error does not affect the decoding process. The `process()` function returns `SUCCESS` and sets the extended error status accordingly.

Table 4-2. MPEG2 Decoder Fatal Error Statuses

| Group or Enumeration Class | Symbolic Constant Name | Value | Description or Evaluation |
|----------------------------|-------------------------|-------|---|
| MPEG2VDEC_ERROR | MPEG2VDEC_ERROR_Failure | 1 | <ul style="list-style-type: none"> ❑ Decoding failed. ❑ Not supported in this version of MPEG2 Decoder. |

Table 4-3. MPEG2 Decoder Non-Fatal Bit stream Error Statuses

| Group or Enumeration Class | Symbolic Constant Name | Value | Description or Evaluation |
|----------------------------|---|-------|---|
| IMPEGDEC_ErrorStatus | MPEG2VDEC_ERROR_bitstream_Overrun | 2 | Decoder has exhausted the given bit stream and run past available number of bits. This indicates either erroneous bit stream or insufficient data to decoder. |
| | MPEG2VDEC_ERROR_unsupported_pictureSpatialScalableExtension | 3 | Unsupported picture spatial scalable extension |

| Group or Enumeration Class | Symbolic Constant Name | Value | Description or Evaluation |
|----------------------------|---|-------|---|
| IMPEGDEC_ErrorStatus | MPEG2VDEC_ERROR_unSupportedPictureTemporalScalableExtension | 4 | Unsupported picture temporal scalable extension |
| | MPEG2VDEC_ERROR_erroneousBitstream | 5 | Erroneous bit-stream |
| | MPEG2VDEC_ERROR_sliceVerticalPosition | 6 | Slice position is outside the permissible range, and hence error in decoding |
| | MPEG2VDEC_ERROR_corruptedHeader | 7 | Corrupted frame header |
| | MPEG2VDEC_ERROR_unsupportedInput | 8 | Unsupported input stream |
| | MPEG2VDEC_ERROR_incorrectWidthHeight | 9 | Width and height of the input sequence is not in accordance with the maximum width and height |
| | MPEG2VDEC_ERROR_nullOutputBuffers | 10 | Output buffer pointers passed by the application are NULL |
| | MPEG2VDEC_ERROR_brokenLinkSet | 11 | Frame cannot be decoded properly as the reference frame that is used for prediction is not available due to the action of editing |
| | MPEG2VDEC_ERROR_droppedFrame | 12 | Frame dropped and is not decoded. See Note for details. |
| | MPEG2VDEC_ERROR_noValidReferences | 13 | Frame not decoded as the valid references required to decode it are unavailable due to dropped frame |
| | MPEG2VDEC_ERROR_noRefBufferToFlush | 14 | Reference buffer is not available to flush on calling <code>control()</code> API with FLUSH command |
| | MPEG2VDEC_ERROR_nullInputBufferPtr | 17 | Input buffer pointer passed by the application is NULL |
| | MPEG2VDEC_ERROR_insufficientOutputBufSize | 18 | Output buffer size(s) provided by the application is not sufficient |
| | MPEG2VDEC_ERROR_nullPointer | 19 | NULL pointer is passed by the application |

| Group or Enumeration Class | Symbolic Constant Name | Value | Description or Evaluation |
|----------------------------|--------------------------------------|-------|---|
| | MPEG2VDEC_ERROR_invalidStructureSize | 20 | Size of XDM structure(s) passed by the application is invalid |

Table 4-4. MPEG2 Decoder Non-Fatal Error Statuses

| Group or Enumeration Class | Symbolic Constant Name | Value | Description or Evaluation |
|----------------------------|--|-------|--|
| IMPEGDEC_ErrorStatus | MPEG2VDEC_ERROR_insufficientUserDataBuffer | 15 | Size of the user buffer provided by the application is not sufficient to completely parse the user data. |
| | MPEG2VDEC_ERROR_nullDisplayHdrBufPtrs | 16 | Display header pointers namely user data, sequence display extension and picture display extension passed by the application are NULL. |
| | MPEG2VDEC_ERROR_invalidDisplayHdrSizes | 21 | Size of display header structure(s) namely user data, sequence display extension, and picture display extension passed by the application are invalid. |

Note:

The error MPEG2VDEC_ERROR_droppedFrame is applicable only for the trick play feature. Decoder returns -1 (IALG_EFAIL) when this error is set. However, the application can continue decoding waiting for the valid frame.

Trick play includes the following three commands:

- ☐ Goto Next I Frame (P and B pictures are dropped)
- ☐ Skip B Frame (B pictures are dropped)
- ☐ Skip Current Frame (I/P/B picture is dropped)

For the non-fatal bit-stream error, MPEG2VDEC_ERROR_sliceVerticalPosition, extended error is set but the decoder returns SUCCESS.

4.2.2 Recommended Steps to Recover from Error

The various categories of MPEG2 Decoder specific errors and the recommended actions to be performed by the application to recover from these errors are listed in this section.

- ❑ In case of errors, the decoder returns current erroneous frame as NULL. The application can call `control()` API with `FLUSH` command to retrieve the valid reference buffer stored inside the decoder.
- ❑ In case of erroneous B frames, the application should not call `control()` API with `FLUSH` command if application intends to continue decoding.
- ❑ If the flags `goto_next_I_frame` or `reverse_play` of `IMPEG2VDEC_DynamicParams` structure is set to 1, the application should not call `control()` API with `FLUSH` command.
- ❑ If application calls `control()` API with `FLUSH` command and references are not available to flush, decoder returns error `MPEG2VDEC_ERROR_noRefBufferToFlush`. This error will overwrite any previous extended error, if exists.
- ❑ On encountering the end of sequence code, decoder returns the last valid reference buffer locked inside. If the end of sequence is not present, the application can retrieve the last valid reference frame stored inside the decoder using the `control()` API with `FLUSH` command.
- ❑ If end of sequence code is encountered and application calls `control()` API with `FLUSH` command, decoder returns error `MPEG2VDEC_ERROR_noRefBufferToFlush` as no valid references are available to flush.
- ❑ At end of sequence, all the parameters of `IMPEG2VDEC_Status` and `outArgs` structure retain the previous values except `frameIdentifier`, `frame_num`, `displayBufs` and `outputID` parameters that represent the last valid reference frame that is output by the decoder in display order.
- ❑ Both in case of error or non-error, to extract the information from the buffers maintained inside the decoder, the structure `IMPEG2VDEC_Buffer_Entry` can be used. This structure indicates the accepted buffer (Y) address and ID that is output by the decoder in display order at some point of time. The number of valid reference buffers stored is given by the `valid_buff_entries` parameter.
- ❑ The `inbuf_status` field in `IMPEG2VDEC_Status` indicates the status of the input buffers passed by the application to the decoder.
- ❑ The value of `inbuf_status` field indicates:
 - **0 - Under Progress:** The buffer is being used by the decoder but not yet accepted. This happens for field pictures. Decode is called twice to generate one frame output. The status of the buffer accepted or rejected is determined at the end of two decode calls.

Hence, when the first call to decode returns `inbuf_status` value as 0, the same buffer should be passed again.

- **1 - Accepted:** Buffer is accepted.
- **-1 - Rejected:** Buffer is rejected. The decoder rejects the buffer on encountering end of sequence, `XDM_PARSE_HEADER`, or an error.

Note:

- ❑ For non-erroneous field pictures, if the input ID of the second decode call is different from the input ID of the first, `inbuf_status` field contains value `-1` (Rejected) during the second decode call. This indicates that the second buffer is rejected. However, the decoder will output the first buffer in display order.
- ❑ If the application finds skipped frames, it may continue to repeatedly display the last buffer given out by the decoder skipped frame number of times. The output buffer descriptor contains the address of the next buffer to be displayed in sequence after the skipped frames.
- ❑ If the following non-fatal errors are found, the application should take necessary corrective action before decoding further:
 - `MPEG2VDEC_ERROR_nullInputBufPtr`
 - `MPEG2VDEC_ERROR_nullOutputBufPtrs`
 - `MPEG2VDEC_ERROR_insufficientoutBufSize`
 - `MPEG2VDEC_ERROR_incorrectWidthHeight`
- ❑ The mandatory corrective actions could be to pass valid input/output buffer pointer(s) or output buffer size(s) or modify the maximum height and width according to the maximum resolution (D1, HDTV_720p, HDTV_1080I and so on) supported.
- ❑ For non-multiple of 16 streams, the decoded output is provided by the decoder in multiple of 16 resolution. Patch is present in the decoded stream that is cropped by the application. The decoder does not do any padding. Hence, the patch contains invalid data.

4.3 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.

4.3.1 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

4.3.1.1 *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 |

4.3.1.2 *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_Sungl eBufDesc | Input | Array of buffer descriptor |

4.3.1.3 *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 buffer in bytes |

4.3.1.4 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 buffer in bytes |
| accessMask | XDAS_Int32 | Output | If the buffer was not accessed by the algorithm processor (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 |

4.3.1.5 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:

For MPEG2 Decoder, the buffer details are:

- ❑ Number of input buffer required is 1
- ❑ Number of output buffer required is 1 for YUV 422ILE and 3 for YUV420P

- ❑ There is no restriction on input buffer size except that it should contain atleast one frame of encoded data.
- ❑ The output buffer address needs to be 128 bit aligned for OMAP 3530.
- ❑ Padding of 128 bytes of zeroes should be done by the application at the end of the input buffer for decoding error streams. This is recommended to avoid decoder from accessing beyond numBytes provided, due to corrupted bitstream.
- ❑ The output buffer sizes (in bytes) for worst case HDTV_1080I format are:

For YUV 420P:

Y buffer = 1920 * 1080

U buffer = 960 * 544

V buffer = 960 * 544

For YUV 422ILE:

Buffer = 1920 * 1080 * 2

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

4.3.1.6 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. It is always returned in number of bytes. |
| 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 |

| Field | Datatype | Input/ Output | Description |
|--------------|------------|------------------|--|
| frameStatus | XDAS_Int32 | Input | Frame status of IVIDEO_Output. It shows the status of the output buffers. Default value is IVIDEO_FRAME_NO_ERROR, which indicates that the output buffer is available. |
| repeatFrame | XDAS_Int32 | Input | Number of times the display process needs to repeat the display progressive frame |
| contentType | XDAS_Int32 | Input | Content type of the buffer |
| chromaFormat | XDAS_Int32 | Input | XDM_ChromaFormat. |

4.3.1.7 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 <i>TMS320 DSP Algorithm Standard API Reference</i> (literature number SPRU360). |
| *process | XDAS_Int32 | Input | Pointer to the <code>process()</code> function |
| *control | XDAS_Int32 | Input | Pointer to the <code>control()</code> function |

4.3.1.8 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 | <code>XDAS_Int32</code> | Input | Size of the basic or extended (if being used) data structure in bytes. |
| maxHeight | <code>XDAS_Int32</code> | Input | Maximum video height to be supported in pixels Default value is 1088. |
| maxWidth | <code>XDAS_Int32</code> | Input | Maximum video width to be supported in pixels Default value is 1920. |
| maxFrameRate | <code>XDAS_Int32</code> | Input | Maximum frame rate in fps * 1000 to be supported. |
| maxBitRate | <code>XDAS_Int32</code> | 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 | <code>XDAS_Int32</code> | Input | Endianness of input data. See <code>XDM_DataFormat</code> enumeration for details. Default value is <code>XDM_BYTE</code> . |
| forceChromaFormat | <code>XDAS_Int32</code> | Input | Sets the output to the specified format. For example, if the output should be in YUV 4:2:2 interleaved (little endian) format, set this field to <code>XDM_YUV_422ILE</code> . See <code>XDM_ChromaFormat</code> enumeration for details. Default value is <code>XDM_YUV_422ILE</code> . |

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 HDTV_1080I format).

4.3.1.9 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 | <code>XDAS_Int32</code> | Input | Size of the basic or extended (if being used) data structure in bytes. |
| decodeHeader | <code>XDAS_Int32</code> | Input | Number of access units to decode: <input type="checkbox"/> 0 (<code>XDM_DECODE_AU</code>) - Decode entire frame including all the headers <input type="checkbox"/> 1 (<code>XDM_PARSE_HEADER</code>) - Decode only one NAL unit Default value is 0. |
| displayWidth | <code>XDAS_Int32</code> | Input | If the field is set to: <input type="checkbox"/> 0 - Uses decoded image width as pitch <input type="checkbox"/> If any other value greater than the decoded image width is given, then this value in pixels is used as pitch. Default value is 0. |
| frameSkipMode | <code>XDAS_Int32</code> | Input | Frame skip mode. See <code>IVIDEO_FrameSkip</code> enumeration for details. |
| frameOrder | <code>XDAS_Int32</code> | Input | Frame display order |
| newFrameFlag | <code>XDAS_Int32</code> | Input | Flag to indicate that the algorithm should start a new frame. Valid values are <code>XDAS_TRUE</code> and <code>XDAS_FALSE</code> . 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 | <code>XDAS_Int32</code> | Input | Flag to indicate that the algorithm should generate MB Data in addition to decoding the data |

Note:

- ☐ Frame skip is not supported. Set the `frameSkipMode` field to `IVIDEO_NO_SKIP`.
- ☐ MPEG2 Decoder will always set the output frames in the display order.
- ☐ MPEG2 Decoder does not support `newFrameFlag` in this version.
- ☐ MPEG2 Decoder does not support `mbDataFlag` in this version.

4.3.1.10 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 (example, B frames). If there is no re-ordering, outputID field in the IVIDDEC2_OutArgs data structure will be same as inputID field. |

Note:

For B-frames, MPEG2 Decoder copies the current `inputID` value to the `outputID` value of `IVIDDEC2_OutArgs` structure. However, for I and P frames, `inputID` value of the previous reference frame is copied to the `outputID` value of `IVIDDEC2_OutArgs` structure.

MPEG2 Decoder copies the `inputID` value to the `outputID` value of `IVIDDEC2_OutArgs` structure after factoring in a display delay of 1.

4.3.1.11 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 <code>XDM_ErrorBit</code> enumeration for details. |
| data | <code>XDM_SingleBufDesc</code> | Output | Buffer Information structure for Information passing buffer |
| maxNumDisplayBufs | XDAS_Int32 | Output | The maximum number of buffers required by the coded |
| 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 <code>IVIDEO_ContentType</code> enumeration for details. |
| outputChromaFormat | XDAS_Int32 | Output | Output chroma format. See <code>XDM_ChromaFormat</code> enumeration for details. |
| bufInfo | <code>XDM_AlgoBufInfo</code> | Output | Input and output buffer information. See <code>XDM_AlgoBufInfo</code> data structure for details. |

Note:

The output chroma format for YUV 4:2:2 interleaved is as shown:

```

u00  y00  v00  y01  u02  y02  v02  y03  .  .
u00  y10  v00  y11  u02  y12  v02  y13  .  .
u20  y20  v20  y21  u22  y22  v22  y23  .  .
u20  y30  v20  y31  u22  y32  v22  y33  .  .
u40  y40  v40  y41  u42  y42  v42  y43  .  .
u40  y50  v40  y51  u42  y52  v42  y53  .  .
u60  y60  v60  y61  u62  y62  v62  y63  .  .
u60  y70  v60  y71  u62  y72  v62  y73  .  .
.    .    .    .    .    .    .    .    .    .

```

- ❑ Algorithm sets the `frameRate` and `bitRate` fields to default values.
- ❑ MPEG2 Decoder will not be using the buffer descriptor meant for passing additional information between the application and the decoder.

4.3.1.12 IVIDDEC2_OutArgs

|| Description

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

|| Fields

| Field | Datatype | Input/ Output | Description |
|---|-----------------------------------|------------------|--|
| <code>size</code> | <code>XDAS_Int32</code> | Input | Size of the basic or extended (if being used) data structure in bytes. |
| <code>bytesConsumed</code> | <code>XDAS_Int32</code> | Output | Bytes consumed per decode call |
| <code>outputID[XDM_MAX_IO_BUFFERS]</code> | <code>XDAS_Int32</code> | Output | Output ID corresponding to <code>displayBufs</code> . A value of zero(0) indicates an invalid ID. The first zero entry in an array will indicate end of valid output IDs within the array. Hence, the application can stop reading the array, when it encounters the first zero entry. |
| <code>decodedBufs</code> | <code>IVIDEO1_Bu fDesc</code> | 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 <code>outBufsInUseFlag</code> , 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. |

| Field | Datatype | Input/ Output | Description |
|-------------------------------------|--------------------|------------------|---|
| 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 |
| free-BufID[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. |

Note:

For B-frames, MPEG2 Decoder copies the current inputID value to the outputID value of IVIDDEC2_OutArgs structure. However, for I and P frames, inputID value of the previous reference frame is copied to the outputID value of IVIDDEC2_OutArgs structure.

4.3.2 MPEG2 Decoder Data Structures

This section includes the following MPEG2 Decoder specific extended data structures:

- ❑ IMPEG2VDEC_Params
- ❑ IMPEG2VDEC_DynamicParams
- ❑ IMPEG2VDEC_InArgs
- ❑ IMPEG2VDEC_Status
- ❑ IMPEG2VDEC_OutArgs

4.3.2.1 IMPEG2VDEC_Params

|| Description

This structure defines the creation parameters and any other implementation specific parameters for the 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. |

4.3.2.2 IMPEG2VDEC_DynamicParams

|| Description

This structure defines the run-time parameters and any other implementation specific parameters for the MPEG2 Decoder 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 IVIDDEC2_DynamicParams data structure for details. |

| Field | Datatype | Input/ Output | Description |
|---------------------|-------------|------------------|---|
| ppNone | XDAS_Int32 | Input | <ul style="list-style-type: none"> ❑ 1 - Indicates no post processing is done on decoded output and the output will always be in 4:2:0 planar format. The decoder reference buffers are exposed as output buffers. Hence instead of output buffers being passed to algorithm, it returns reference buffers. ❑ 0 - Indicates post processing can be done on the decoded output. Algorithm uses the application provided display buffers for outputting decoded data. Default value is 0. |
| dyna_chroma_format | XDAS_Int32 | Input | <p>Sets the output to the specified format at the frame level. To use this feature, <code>forceChromaFormat</code> of <code>IVIDDEC2_Params</code> structure should be set to <code>XDM_YUV_422ILE</code>.</p> <p>See <code>XDM_ChromaFormat</code> enumeration for details. Default value is <code>XDM_YUV_422ILE</code>.</p> |
| displayFieldReorder | XDAS_UInt32 | Input | <ul style="list-style-type: none"> ❑ 1 - Reorder the bottom field in case of non-progressive sequences where bottom field is sent ahead of top field for frame pictures. ❑ 0 - No reordering. Default value is 0. |
| frameLevelByteSwap | XDAS_UInt32 | Input | <ul style="list-style-type: none"> ❑ 1 - Enable byte swap inside the decoder only for the current frame number of bytes. This flag restricts the byte swap at the frame level for optimization. ❑ 0 - Byte swap the input buffer up to the <code>numBytes</code> provided by the application. Default value is 0. |
| no_delay_display | XDAS_UInt32 | Input | <ul style="list-style-type: none"> ❑ 1 - Decoded output of the first I frame of the sequence is given back immediately to the application for display. ❑ 0 - Normal operation. There is one frame delay before providing the first I frame to the application. Default value is 0. |
| goto_next_I_frame | XDAS_UInt32 | Input | <ul style="list-style-type: none"> ❑ 1 - Only I frames are decoded until the flag is reset. ❑ 0 - All the frames are decoded. Default value is 0. |

| Field | Datatype | Input/ Output | Description |
|-------------------|-------------|------------------|---|
| skip_B_frame | XDAS_UInt32 | Input | <input type="checkbox"/> 1 - B frames are skipped until the flag is reset. <input type="checkbox"/> 0 – B frames are decoded. Default value is 0 |
| skip_curr_frame | XDAS_UInt32 | Input | <input type="checkbox"/> 1 - Current frame is skipped until the flag is reset. <input type="checkbox"/> 0 - No skip. Default value is 0. |
| seek_frame_end | XDAS_UInt32 | Input | Controls the behaviour of the decoder in case of dropped frame. <input type="checkbox"/> 1 - Decoder seeks to the end of the frame, updates bytes consumed accordingly and returns to the application <input type="checkbox"/> 0 - Decoder returns with bytes consumed set to zero. |
| getDisplayHdrInfo | XDAS_UInt32 | Input | <input type="checkbox"/> 1 - Decoder exposes the headers user data, sequence display extension, and picture display extension. <input type="checkbox"/> 0 - Display headers are not exposed. Default value is 0. |
| reverse_play | XDAS_UInt32 | Input | <input type="checkbox"/> 1 - Decoder uses the reference buffers that the application provides for reconstruction instead of the internal reference buffers. This command starts the decoder in reverse play mode. <input type="checkbox"/> 0 - Normal decoding. Internal reference buffers are used for reconstruction. Default value is 0. |
| robustness_level | XDAS_UInt32 | Input | Indicates level of robustness of the decoder. <input type="checkbox"/> 1 - Current level of robustness <input type="checkbox"/> 0 - Robustness level is less with the risk of little system instability Default value is 1. |

Note:

- ☐ If `dyna_chroma_format` is not used, it should always be set same as `forceChromaFormat` of `IVIDDEC2_Params` structure.
- ☐ If `dyna_chroma_format` is used, `GETBUFINFO` should be invoked after calling `SETPARAMS` to get the buffer allocation information accordingly.
- ☐ If `ppNone = 1`, `forceChromaFormat` of `IVIDDEC2_Params` structure and `dyna_chroma_format` of `IMPEG2VDEC_DynamicParams` structure should always be set to `XDM_YUV_420P`.
- ☐ If `ppNone = 0`, the post processing that can be done on the decoded output is conversion from `XDM_YUV_420P` to

```
XDM_YUV_422ILE
```

- ❑ The flag `seek_frame_end` should always be set to 1, if one of the flags `goto_next_I_frame`, `skip_B_frame`, or `skip_curr_frame` is set to 1.
- ❑ Flag `no_delay_display` should always be set to 1, if `goto_next_I_frame` or `reverse_play` is set to 1.
- ❑ For reverse play, reference buffers should be provided by the application in 420 planar format only. In other words, decoder requires three forward reference buffers and three backward reference buffers for reconstruction.
- ❑ For reverse play ,reference buffers should be provided by `XDM1_BufDesc (inBufs)`.

4.3.2.3 IMPEG2VDEC_InArgs

|| Description

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

|| Fields

| Field | Datatype | Input/ Output | Description |
|---------------------------|------------------------------|------------------|--|
| <code>viddecInArgs</code> | <code>IVIDDEC2_InArgs</code> | Input | See <code>IVIDDEC2_InArgs</code> data structure for details. |

4.3.2.3.1 IMPEG2VDEC_Sequence_Display_Extension

|| Description

This structure defines the sequence level display information.

|| Fields

| Field | Datatype | Input/ Output | Description |
|-------------------------------|-------------------------|------------------|--|
| <code>video_format</code> | <code>XDAS_Int32</code> | Output | Representation of the pictures before being coded like NTSC, PAL and so on |
| <code>display_hor_size</code> | <code>XDAS_Int32</code> | Output | Display horizontal size |
| <code>display_ver_size</code> | <code>XDAS_Int32</code> | Output | Display vertical size |

4.3.2.3.2 IMPEG2VDEC_Picture_Display_Extension

|| Description

This structure defines the picture level display information.

|| Fields

| Field | Datatype | Input/ Output | Description |
|---------------------|------------|------------------|---|
| *frm_cen_hor_offset | XDAS_Int32 | Output | Pointer to array of horizontal offsets |
| *frm_cen_ver_offset | XDAS_Int32 | Output | Pointer to array of vertical offsets |
| frm_cen_offsets | XDAS_Int32 | Output | Number of frame center offsets present. The maximum value is 3. |

Note:

- ❑ Application has to allocate memory for the structures `User_Data`, `IMPEG2VDEC_Sequence_display_extension`, and `IMPEG2VDEC_Picture_display_extension` using `XDM_BufDesc outBufs`.
- ❑ The updated structures are provided by the decoder using extended structure `XDM_BufDesc outBufs`.
- ❑ Both horizontal offsets and vertical offsets are in units of 1/16th sample.
- ❑ The number of horizontal and vertical offsets depends on the value `frm_cen_offsets`.

4.3.2.3.3 IMPEG2VDEC_Buffer_Entry

|| Description

This structure provides the buffer address accepted and held by the decoder.

|| Fields

| Field | Datatype | Input/ Output | Description |
|---------------|-------------|------------------|---|
| inBuf_address | XDAS_UInt32 | Output | Accepted buffer (Y) address that will be output by the decoder in display order |
| inputID | XDAS_Int32 | Output | Input ID of the accepted buffer |

4.3.2.4 IMPEG2VDEC_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 |
|---|--------------------------------------|------------------|---|
| <code>viddecStatus</code> | <code>IVIDDEC2_Status</code> | Output | See <code>IVIDDEC2_Status</code> data structure for details |
| <code>displayBufinfo</code> | <code>IVIDEO1_BufDesc</code> | Output | Buffer information for current displayable frame. See <code>IVIDEO1_BufDesc</code> data structure for details. If the <code>control()</code> API is called with <code>FLUSH</code> command, decoder returns the buffer information for the valid reference frame stored inside the decoder. |
| <code>outputID</code> | <code>XDAS_Int32</code> | Output | See <code>IVIDDEC2_OutArgs</code> data structure for details. If the <code>control()</code> API is called with <code>FLUSH</code> command, this ID indicates the valid reference frame stored inside the decoder. |
| <code>acceptedBufs[XDM_MAX_IO_BUFFERS]</code> | <code>IMPEG2VDEC_Buffer_Entry</code> | Output | Accepted buffer addresses stored internally by the decoder |
| <code>valid_buff_entries</code> | <code>XDAS_UInt32</code> | Output | Valid number of structures in the array <code>acceptedBufs</code> |
| <code>non_vcl_length_at_start</code> | <code>XDAS_Int32</code> | Output | Length of non Video Coding Layer (VCL) data per frame |
| <code>frame_num</code> | <code>XDAS_Int32</code> | Output | Frame number of the decoded picture in the display order. This parameter is parsed from the bit-stream. It corresponds to the field <code>temporal_reference</code> of the Picture Header. |
| <code>frameIdentifier</code> | <code>XDAS_UInt32</code> | Output | Timestamp of the frame returned by the decoder in the display order. This parameter is set based on the input ID. |
| <code>pict_struct</code> | <code>ePicStrFormat</code> | Output | Indicates decoded picture type. In case of field picture, the top or bottom field can be identified by looking at this. See Table 4-1 (<code>ePicStrFormat</code>) for details. |
| <code>video_format</code> | <code>XDAS_Int32</code> | Output | Indicates representation of the pictures |
| <code>display_hor_size</code> | <code>XDAS_Int32</code> | Output | Display horizontal size |
| <code>display_ver_size</code> | <code>XDAS_Int32</code> | Output | Display vertical size |

| Field | Datatype | Input/ Output | Description |
|----------------------|------------|------------------|--|
| actualWidth | XDAS_Int32 | Output | Actual width in pixels |
| actualHeight | XDAS_Int32 | Output | Actual height in pixels |
| new_user_data | XDAS_Int8 | Output | Set to 1, if user data is parsed from the current frame |
| new_seq_display | XDAS_Int8 | Output | Set to 1, if sequence display extension is parsed from the current frame |
| new_pict_display | XDAS_Int8 | Output | Set to 1, if picture display extension is parsed from the current frame |
| is_mpeg2 | XDAS_Int8 | Output | Indicates MPEG2 streams. This field is set to zero for MPEG1 streams. |
| end_of_seq | XDAS_Int8 | Output | End of sequence flag, may be used for displaying the last frame stored in buffer. |
| stepSize | XDAS_Int8 | Output | Quantization step size used in the frame |
| progressive_frame | XDAS_Int8 | Output | Frame type: <input type="checkbox"/> 1 – Progressive <input type="checkbox"/> 0 - Interlaced |
| progressive_sequence | XDAS_Int8 | Output | Sequence type: <input type="checkbox"/> 1 - Progressive <input type="checkbox"/> 0 - Non-progressive |
| closed_gop | XDAS_Int8 | Output | B-pictures encoded using only backward prediction or intra coding |
| broken_link | XDAS_Int8 | Output | B-pictures cannot be correctly decoded |
| aspect_ratio | XDAS_Int8 | Output | Aspect ratio information |
| inbuf_status | XDAS_Int8 | Output | Status of the display buffer passed by the application to the decoder: <input type="checkbox"/> 0 - Under progress <input type="checkbox"/> 1 - Buffer is accepted <input type="checkbox"/> -1 - Buffer is rejected |

Note:

- ☐ If the `control()` API is called with `FLUSH` command, decoder updates only the fields `extendedError` of the structure `IVIDDEC2_Status`, `displayBufinfo`, and `OutputID` of the structure `IMPEG2VDEC_Status`.
- ☐ The `extended status` structure variables `video_format`, `display_hor_size`, `display_ver_size` will get updated only when `getDisplayHdrInfo` of `DynamicParams` structure becomes 1.

4.3.2.5 IMPEG2VDEC_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. |

4.4 Interface Functions

This section describes the Application Programming Interfaces (APIs) used in the MPEG2 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).

4.4.1 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()
```

4.4.2 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()`

4.4.3 Control API

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

|| Name

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

|| Synopsis

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

|| Arguments

```
IVIDDEC2_Handle handle; /* algorithm instance handle */
IVIDDEC2_Cmd id; /* algorithm specific control commands*/
IVIDDEC2_DynamicParams *params /* algorithm run time
parameters */
IVIDDEC2_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 `IVIDDEC2_DynamicParams` and `IVIDDEC2_Status` data structures respectively.

Note:

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

|| Preconditions

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

- ❑ `control()` can only be called after a successful return from `algInit()` and `algActivate()`.
- ❑ If algorithm uses DMA resources, `control()` can only be called after a successful return from `DMAN3_init()`.
- ❑ `handle` must be a valid handle for the algorithm's instance object.

|| Postconditions

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

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

|| Example

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

|| See Also

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

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

|| Arguments

```
IVIDDEC2_Handle handle; /* algorithm instance handle */

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

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

IVIDDEC2_InArgs *inargs /* algorithm runtime input
arguments */

IVIDDEC2_OutArgs *outargs /* algorithm runtime output
arguments */
```

|| Return Value

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

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

|| Description

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

The second and third arguments are pointers to the input and output buffer descriptor data structures respectively (see `XDM1_BufDesc` and `XDM_BufDesc` data structure for details).

The fourth argument is a pointer to the `IVIDDEC2_InArgs` data structure that defines the run time input arguments for an algorithm instance object.

The last argument is a pointer to the `IVIDDEC2_OutArgs` data structure that defines the run time output arguments for an algorithm instance object.

Note:

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

|| Preconditions

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

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

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

|| Postconditions

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

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

|| Example

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

|| See Also

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

Note:

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

| | |
|------------------------|--|
| Name | <code>algDeactivate()</code> – save all persistent data to non-scratch memory |
| Synopsis | |
| Arguments | <code>Void algDeactivate(IALG_Handle handle);</code> |
| Return Value | <code>IALG_Handle handle; /* algorithm instance handle */</code> |
| Description | <p><code>Void</code></p> <p><code>algDeactivate()</code> saves any persistent information to non-scratch buffers using the persistent memory that is part of the algorithm's instance object.</p> <p>The first (and only) argument to <code>algDeactivate()</code> 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 <code>algActivate()</code> and processing.</p> <p>For more details, see <i>TMS320 DSP Algorithm Standard API Reference</i> (literature number SPRU360).</p> |
| See Also | <code>algActivate()</code> |

4.4.5 Termination API

Termination API is used to terminate the algorithm instance and free up the memory space that it uses.

|| Name

`algFree()` – determine the addresses of all memory buffers used by the algorithm

|| Synopsis

```
XDAS_Int32 algFree(IALG_Handle handle, IALG_MemRec  
memTab[]);
```

|| Arguments

```
IALG_Handle handle; /* handle to the algorithm instance */  
IALG_MemRec memTab[]; /* output array of memory records */
```

|| Return Value

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

|| Description

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

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

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

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

|| See Also

`algAlloc()`

This page is intentionally left blank

Frequently Asked Questions

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

Table 5-1. FAQs for MPEG2 Decoder on C64x+

| Question | Answer |
|---|---|
| Does the decoder supports reverse play mode? | Yes. The decoder supports reverse play as well as trick play. It uses the reference buffers provided by the application for reconstruction instead of the internal reference buffers. Reverse play can be started by setting <code>reverse_play</code> field of <code>IMPEG2VDEC_DynamicParams</code> to 1. |
| Does the decoder support non-multiple of 16 height and width? | The decoder supports decoding of bit-streams with non-multiple of 16 frame sizes. The output buffer in this case can be of the actual frame size. The height and width value used to create decoder instance should be multiple of 16. |
| Does the decoder support MPEG-1 Constrained Parameters Bit-streams? | Yes. The decoder supports MPEG-1 Constrained Parameters Bit-streams. |
| What are the output formats supported by the decoder? | MPEG2 Decoder supports YUV 420 planar and YUV 422 interleaved output formats. |
| What is the maximum resolution supported by MPEG2 Decoder? | MPEG2 Decoder can decode maximum HDTV_1080I resolution video streams. |

This page is intentionally left blank

Revision History

This user guide revision history highlights the changes made to the SPRUEA4B codec specific user guide to make it SPRUEA4C.

Table A-1. Revision History of MPEG2 Decoder on C64x+

| Section | Addition/Modification/Deletion |
|-------------|-------------------------------------|
| Section 2.2 | ❑ Modified top-level directory name |
| Section 2.5 | ❑ Modified sample Testvecs.cfg file |