MPEG4 Restricted Simple Profile Decoder Codec on DM355

User's Guide

Literature Number: SPRUFE3C

November 2007 – Revised April 2008

Contents

List of Figures

List of Tables

Read This First

This document describes how to install and work with Texas Instruments' (TI) MPEG4 simple profile decoder implementation on the DM355 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 DM355 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.

Related Documentation From Texas Instruments

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

- *TMS320 DSP Algorithm Standard API Reference* (SPRU360) describes all the APIs that are defined by the *TMS320 DSP Algorithm Interface Standard* (also known as XDAIS) specification.
- *Technical Overview of eXpressDSP* - Compliant Algorithms for DSP Software Producers (SPRA579) describes how to make algorithms compliant with the *TMS320 DSP Algorithm Standard* which is part of TI's eXpressDSP technology initiative.
- *xDAIS-DM (Digital Media) User Guide* (SPRUEC8)
- *Using DMA with Framework Components for C64x+* (SPRAAG1)

Related Documentation

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

- *ISO/IEC 14496-2:2004, Information Technology -- Coding of Audio-Visual Objects -- Part 2: Visual (Approved in 2004-05-24)*
- *H.263 ITU-T Standard – Video Coding for Low Bit Rate Communication*

Abbreviations

The following abbreviations are used in this document:

Table 1. List of Abbreviations

Abbreviation	Description	Abbreviation	Description
API	Application programming interface	Kbps	Kilo bits per second
CBR	Constant bit rate	MPEG	Motion Picture Expert Group
CCS	Code Composer Studio	NTSC	National Television Standards Committee
CIF	Common intermediate format	OBMC	Overlapped block motion compensation
DVD	Digital Versatile Disc	SXVGA	Super Extended video graphics array
EVM	Evaluation module	UMV	Unrestricted motion vector
fps	Frames per second	VBR	Variable bit rate
GOB	Group of Block	VOP	Video object plane
HD	High Definition	XDAIS	eXpress DSP Algorithm Interface Standard
IDMA3	DMA Resource specification and negotiation protocol	XDM	eXpressDSP Digital Media

Text Conventions

The following conventions are used in this document:

- Text inside back-quotes (‘‘) represents pseudo-code.
- Program source code and command line commands are shown in a mono-spaced font.
- Parameters are shown in italics.

Support

When contacting TI for support on this codec, please quote the product name (MPEG4 Simple Profile Decoder on DM355) and version number. The version number of the codec is included in the Title of the Release Notes that accompanies this codec.

Trademarks

Code Composer Studio and eXpressDSP are trademarks of Texas Instruments.

All trademarks are the property of their respective owners.

Software Copyright

Software Copyright 2008 Texas Instruments Inc.

Chapter 1

SPRUFE3C – November 2007 – Revised April 2008

Introduction

This chapter provides a brief introduction to XDAIS, XDM and IDMA3. It also provides an overview of TI's implementation of the MPEG4 Simple Profile Decoder on the DM355 platform and its supported features.

Topic **Page**

1.1 Overview of XDAIS, XDM and IDMA3

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). IDMA3 is the standard interface to algorithms for DMA resource specification and negotiation protocols. This interface allows the client application to query and provide the algorithm with its requested DMA resources.

1.2 XDAIS Overview

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

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

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

Once an algorithm instance object is created, it can process data in real-time. The algActivate() API provides a notification to the algorithm instance that one or more algorithm processing methods 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* (SPRU360).

1.3 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:

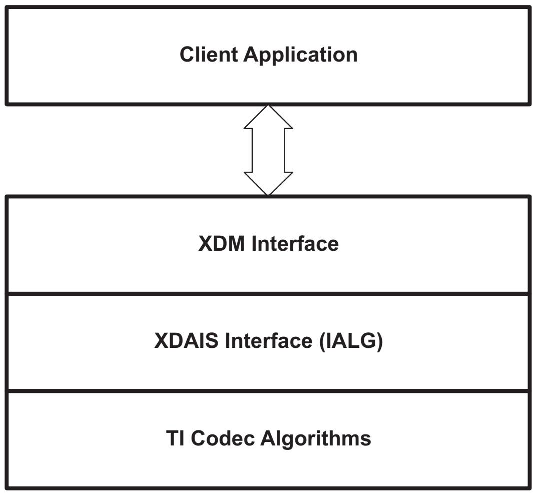
- control()
- process()

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.

Figure 1-1 depicts the XDM interface to the client application.

Figure 1-1. XDM Interface to the Client Application



As depicted in Figure 1-1, 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. As 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, such as H.263, with minimal changes to the client application.

For more details, see *xDAIS-DM (Digital Media) User Guide* (SPRUEC8.

1.4 IDMA3 Overview

Client applications use the algorithm's IDMA3 interface to query the algorithm's DMA resource requirements and grant the algorithm logical DMA resources via handles. Figure 1-1 shows a typical IDMA3 interface implemented by the algorithm module, which would be used by the client applications to query the algorithm's DMA needs. The algorithm specifies the number of separate EDMA/QDMA channels and PaRamsets it requires through memRecs. The IDMA3 standard defines the following APIs:

- dmaChangeChannels()
- dmaGetChannelCnt()
- dmaGetChannels()
- dmaInit()

dmaChangeChannels() Called by an application whenever logical channels are moved at run-time. This allows the application to re-initialize the channel properties whenever allocated resources are not available.

dmaGetChannelCnt() Called by an application to query an algorithm about its number of logical DMA channel requests.

dmaGetChannels() Called by an application to query an algorithm about its DMA channel requests at initialization time, or to get the current channel holdings. Through this API, the algorithm specifies the number of TCCs and PaRamSets it requires and the properties of these resources when called during initialization time.

dmaInit() Called by an application to grant DMA handle(s) to the algorithm at initialization.

For more details, see *Using DMA with Framework Components for C64x+* (SPRAAG1).

1.5 Features and Limitations

1.5.1 Features

- eXpressDSP® Algorithm Interface Standard (XDAIS) compliant
- eXpressDSP Digital Media (xDM) interface and IDMA3 compliant
- Implements IVIDDEC2 interface of xDM
- Supports MPEG4 simple profile levels 0, 1, 2 and 3 with the following limitations:
  – No support for 4 MV
  – No support for MV ranges beyond -32 to 31
  – No support for escape 1 and 2 VLC
  – No support for DP and RVLC
- Can also decode the following formats:
  – VGA (640 x 480)
  – D1 (720 x 480)
  – 720P (1280 x 720)
  – SXVGA (1280 x 960)
- Supports Half Pel Interpolation (HPI) for motion compensation
- Supports 1 motion vector encoding for motion estimation (1MV/MB) with (-32, +31) half pel search range.
- Supports Unrestricted Motion Vectors (UMV).
- Supports streams with DC and AC prediction
- Supports streams with resync marker (RM)
- Supports streams with short video header (SVH)
- Supports YUV 4:2:2 interleaved data as an output
- Supports Display Width feature; i.e., display width can be greater than the image width
- Supports Rotation (0, 90, 180 and 270 degrees) integrated with the Decoder for certain image formats (QVGA (320x240), VGA (640x480), 720P (1280x720) and SXVGA (1280x960)). Also supports rotation of 240x320 (rotated QVGA) and 480x640 (rotated VGA).

- Can decode all DM355 encoded streams
- Can decode streams of VBR, CBR and CVBR rate control
- Supports frame level reentrancy
- Supports multi instance of MPEG4 Decoder, and single/multi instance of MPEG4 Decoder with other DM355 codecs
- Validated on DM355 EVM

1.5.2 Limitations

The limitations will not be removed in future releases. These limitations are not defects, but intentional or known deficiencies.

- The current version of MPEG4 SP decoder is a restricted decoder. It can decode ONLY streams encoded with the DM355 MPEG4 Encoder.
  – Does not support Video packet resynchronization
  – Does not support Data partitioning (DP)
  – Does not support Reversible VLCs (RVLCs)
  – Does not support Header extension code (HEC)
- Does not support 4MV
- Does not support MV range beyond -32 and +31
- Does not support for escape 1 and 2 VLC
- Does not support arbitrary width and height
  – Supports image width as multiple of 16 and height as multiple of 16.
  – Does not support image width below 160
- Does not support decoding of 720x1280 (rotated 720P) and 960x1280 (rotated SXVGA) formats
- Only limited support of IDMA3 interface. Please refer to Section 3.1 for details.

Installation Overview

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

Topic **Page**

2.1 System Requirements

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

2.1.1 Hardware

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

2.1.2 Software

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

- Linux: MontaVista Linux 4.0.1
- Code Generation Tools: This project is compiled, assembled, and linked using the arm_v5t_le-gcc compiler

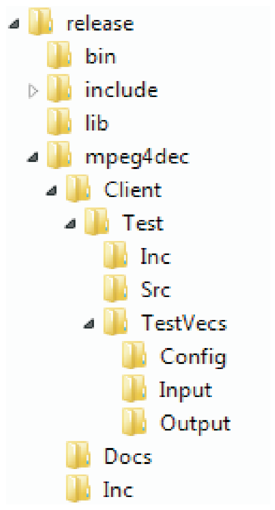
2.2 Installing the Component

To install the codec, follow the instructions in the release notes. The code location is as follows:

MPEG4 Decoder algorithm code is in a directory mpeg4dec placed in DM355Codecs/release.

Figure 2-1 shows the sub-directories structure of mpeg4dec directory.

Figure 2-1. Component Directory Structure



```
release
  bin
  include
  lib
  mpeg4dec
    Client
      Test
        Inc
        Src
        TestVecs
          Config
          Input
          Output
    Docs
    Inc
```

Table 2-1 provides a description of the sub-directories created in the release/mpeg4dec directory.

Table 2-1. Component Directories

Sub-Directory	Description
mpeg4dec/Docs	Contains user guide, datasheet, and release notes
mpeg4dec /Client/Test/Src	Contains application C files
mpeg4dec/Client/Test/Inc	Contains header files needed for the application code
mpeg4dec/Client/Test/TestVecs	Contains test vectors, configuration files
/Include	Contains the include files needed by application and codec.
/lib	Contains MPEG4 Decoder and other support libraries
/bin	Contains MPEG4 Decoder executable “mp4dec”

The DM355 MPEG4 decoder library is put into the /release/lib directory and the xDM headers are put in the /release/include directory.

2.3 Building and Running the Sample Test Application on Linux

The sample test application that accompanies this codec component takes input files and dumps output YUV files as specified in the command line arguments. To build and run the sample test application in Linux, follow these steps:

1. Verify that libmp4vdec.a library is present in the DM355Codecs/release/lib directory.
2. Verify that support libraries (libcosl.a, libdm355.a, libcmem.a) are present in the DM355Codecs/release/lib directory.
3. Change directory to DM355Codecs/release/mpeg4dec/Client/Test/Src and type “make clean” followed by a “make” command. This uses the makefile in that directory to build the test executable mp4dec into the DM355Codecs/release/bin directory.

Note: The ARM tool chain i.e., arm_v5t_le-gcc (ARM gcc) compiler path must be set in user's environment path before building the MPEG4 decoder executable.

4. To run the mp4play executable on the DM355 EVM board, use the following instructions:
   a. Set up the DM355 EVM Board. For information about setting up the DM355 environment, see the *DM355 Getting Started Guide* released in “doc” directory in DVSDK release package.
   b. the MPEG4 Decoder Executable
      i. For running the MPEG4 decoder executable, copy the executable "mp4dec" along with the entire "TestVecs" directory provided with the release package at project/mpeg4dec/Client/Test to the target directory.
      ii. Copy the kernel modules "dm350mmap.ko" and "cmemk.ko" to the target directory. These modules are provided with the release package in project/bin directory.
      iii. Copy "loadmodules.sh" provided with release package at project/bin to the target directory.
      iv. o Execute following commands in sequence to run the MPEG4 decoder executable.

      ```
      $./loadmodules.sh
      $./mp4dec
      ```

      This will run the MPEG4 decoder with base parameters. To run the MPEG4 Decoder with extended parameters, change the config file in Testvecs.cfg to Testparams.cfg (TestVecs/Config/) and execute

      ```
      $./mp4dec –ext
      ```

2.4 Configuration Files

This codec ships with:

- A generic configuration file (Testvecs.cfg) that specifies input file, output yuv file, and parameter file for each test case.
- A Decoder parameter file (Testparams.cfg) that specifies the configuration parameters used by the test application to configure the decoder for specific test cases.
- The MPEG4 decoder has two modes: extended parameters mode and base parameters mode, which can be specified in the command line argument as mentioned previously.

2.4.1 Test Configuration Files

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

The format of the Testvecs.cfg file is:

X
Config
Input
Output

where:

- X may be set as:

```
                          0- for output dumping
```

- *Config* is the Decoder configuration file. For details, see Section 2.4.2.
- *Input* is the input filename (use complete path).
- *Output* is the output filename.

A sample Testvecs.cfg file is as follows:

```
0
./TestVecs/Config/Testparams.cfg
./TestVecs/Input/akiyo_160x128_422.bits
./TestVecs/Output/akiyo_160x128_422.yuv
```

2.4.2 Decoder Configuration file for base parameters

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

A sample Testparams.cfg file is as shown:

```
# New Input File Format is as follows
# <ParameterName> = <ParameterValue> # Comment
#
##########################################################################################
# Parameters
##########################################################################################
maxHeight          = 960   # max height in Pels, must be multiple of 16
maxWidth           = 1280  # max width in Pels, must be multiple of 16
dataEndianness     = 0     # Frame Rate per second*1000 (1-100)
forceChromaFormat = 4     # only 422ILE supported
```

2.4.3 Decoder Configuration file for extended parameters

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.

```
# New Input File Format is as follows
# <ParameterName> = <ParameterValue> # Comment
#
##########################################################################################
# Parameters
##########################################################################################
maxHeight          = 960   # max height in Pels, must be multiple of 16
maxWidth           = 1280  # max width in Pels, must be multiple of 16
dataEndianness     = 0     # Frame Rate per second*1000 (1-100)
forceChromaFormat = 4     # only 422ILE supported
DecRotation        = 0     # 0,1,2,3 corresponds to 0,90,180,270 degrees
displayWidth       = 0     #Display width of application buffer
meRange            = 31    #only 7 and 31 are supported
UMV                = 1     #UMV support
```

Sample Usage

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

3.1 MPEG4 Decoder Client Interfacing Constraints

The following constraints should be taken into account while implementing the client for the MPEG4 decoder library in this release:

- DMA requirements of MPEG4 Decoder: Current implementation of the MPEG4 decoder uses the following TCCs for its DMA resource requirements along with its associated PaRamSets:

Channel Number	Associated PaRamSet Numbers
32 to 63 except 51	32 to 63 except 51(paRamSet number = channel number)

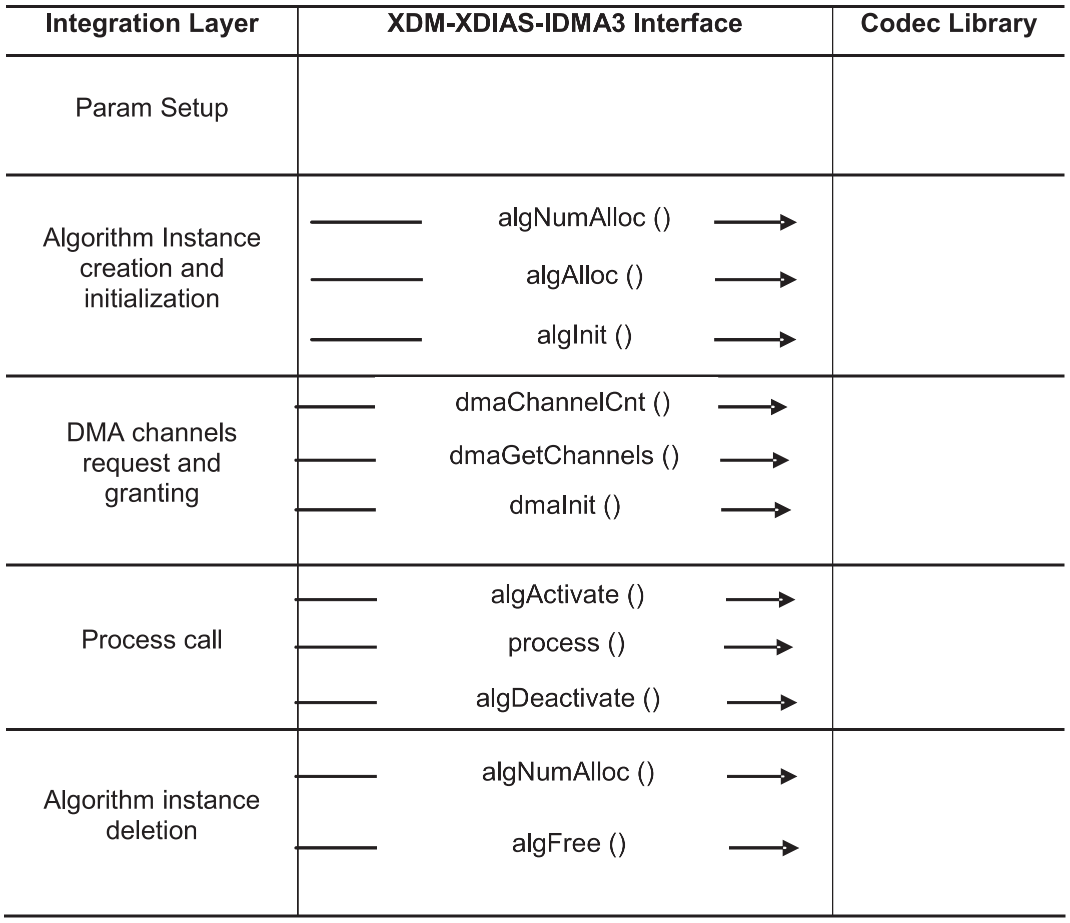
  Apart from these 31 TCCs requirements, it also needs 14 more PaRamSets that are allocated through IDMA3 interface.
- Client application maps all the DMA channels used by MPEG4 decoder to the same queue. This is required for the codec to function normally. The codec does not map channels to queue.
- For multiple instances of a codec and/or different codec combinations, the application can use the same group of channels and PaRAM entries across multiple codecs. The AlgActivate and AlgDeactivate calls made by client application and implemented by the codecs, perform context save/restore to allow multiple instances of same codec and/or different codec combinations.
- As all codecs use the same hardware resources, only one process call per codec should be invoked at a time (frame level reentrancy). The process call needs to be wrapped within activate and deactivate calls for context switch. Refer to XDM specification on activate/deactivate.
- Is there are multiple codecs running with frame level reentrancy, the client application has to perform time multiplexing of process calls of different codecs to meet desired timing requirements between video/image frames.
- The ARM and DDR clock must be set to the required rate for running single or multiple codec
- The codec combinations feasibility is limited by processing time (computational hardware cycles) and DDR bandwidth.
- Codec atomicity is supported at frame level processing only. The process call has to run until completion before another process call can be invoked.

3.2 Overview of the Test Application

The test application exercises the IMP4VDEC_Params extended class of the MPEG4 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

3.2.1 *Parameter Setup*

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

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

1. Sets the IMP4VDEC_Params structure based on the values given in the test application.
2. Reads the input bit stream into the application input buffer.

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

3.2.2 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. Apart from algorithm memory allocation, the application needs to call the IDMA3_Create() function. This function uses the algorithm instance created in the previous call of ALG_create and provides the algorithm with the requisite DMA resources. The following APIs implemented by the algorithm are called in the following sequence:

1. **dmaGetChannelCnt()** - To query the algorithm about the number of memory records it requires. In the present implementation, it always defaults to 1.
2. **dmaGetChannels()** - To query the algorithm about the number of additional PaRamSets it requires in the channel records. In the current implementation, algorithm uses hard-coded channels and its associated TCCs and PaRamSets internally. The client application using the algorithm's IDMA3 interface allocates additional PaRamSets requirements.
3. **dmaInit()** - To initialize the algorithm with continuous PaRamSet addresses allocated to the algorithm during this instance.

3.2.3 Process Call in Single Instance Scenario

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

1. Calls algActivate(), which initializes the decoder state and some hardware memories and registers.
2. Sets the input and output buffer descriptors required for the process() function call.
3. Calls the process() function to decode a single frame of data. The inputs to the process function are input and output buffer descriptors, and the pointer to the IVIDDEC2_InArgs and IVIDDEC2_OutArgs structures. The process() function should be called multiple times to decode multiple frames.
4. Call algDeactivate(), which performs releasing of hardware resources and saving of decoder instance values.
5. process() is made a blocking call, but an internal OS specific layer enables the process to be pending on a semaphore while hardware performs complete MPEG4 Decode
6. Other specific details of the process() function remain the same as described in Section 3.2.6 and constraints described in Section 3.1 are applicable.

Note: The algActivate () call is mandatory before the first process() call, as it does hardware initialization.

3.2.4 Algorithm Instance Deletion

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

1. **algNumAlloc()** - To query the algorithm about the number of memory records it used
2. **algFree()** - To query the algorithm for the memory record information and then free i up for the application

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

3.2.5 Usage in Multiple Instance Scenario

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

- DMA channels and PaRamSets
- MPEG-4-JPEG co-processor and their memory areas

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

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

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

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

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

3.2.6 Process Call with algActivate and algDeactivate

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

1. Sets the input and output buffer descriptors required for the process() function call.
2. Calls algActivate(), which initializes the decoder state and some hardware memories and registers.
3. Calls the process() function to decode a single frame of data. The inputs to the process function are input and output buffer descriptors, and the pointer to the IVIDDEC2_InArgs and IVIDDEC2_OutArgs structures.
4. Calls algDeactivate(), which performs releasing of hardware resources and saving of decoder instance values.
5. Other specific details of the process() function remain the same as described in Section 3.2.3 and constraints described in Section 3.1 are applicable.

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

API Reference

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

Topic **Page**

4.1 Symbolic Constants and Enumerated Data Types

This section summarizes all the symbolic constants specified as either #define macros and/or enumerated C data types. Described alongside the macro or enumeration is the semantics or interpretation of the same in terms of what value it stands for and what it means.

Table 4-1. List of Enumerated Data Types

Group or Enumeration Class	Bit	Symbolic Constant Name	Value	Description or Evaluation
IVIDEO_FrameType		IVIDEO_I_FRAME	0	Intra coded frame
		IVIDEO_P_FRAME	1	Forward inter coded frame
		IVIDEO_B_FRAME	2	Bi-directional inter coded frame. Not supported in this version of MPEG4 decoder.
		IVIDEO_IDR_FRAME	3	Intra coded frame that can be used for refreshing video content. Not supported in this version of MPEG4 decoder.
IVIDEO_ContentType		IVIDEO_PROGRESSIVE	0	Progressive video content.
		IVIDEO_INTERLACED	1	Interlaced video content. Not supported in this version of MPEG4 decoder.
IVIDEO_FrameSkip		IVIDEO_NO_SKIP	0	Do not skip the current frame
		IVIDEO_SKIP_P	1	Skip forward inter coded frame. Not supported in this version of MPEG4 decoder.
		IVIDEO_SKIP_B	2	Skip bi-directional inter coded frame. Not supported in this version of MPEG4 decoder.
		IVIDEO_SKIP_I	3	Skip intra coded frame. Not supported in this version of MPEG4 decoder.
XDM_DataFormat		XDM_BYTE	1	Default value. Big endian stream.
		XDM_LE_16	2	16-bit little endian stream. Not supported in this version of MPEG4 decoder.
		XDM_LE_32	3	32-bit little endian stream. Not supported in this version of MPEG4 decoder.
XDM_ChromaFormat		XDM_YUV_420P	1	YUV 4:2:0 planar Not applicable for MPEG4 decoder.
		XDM_YUV_422P	2	YUV 4:2:2 planar. Not applicable for MPEG4 decoder.
		XDM_YUV_422IBE	3	YUV 4:2:2 interleaved (big endian). Not applicable for MPEG4 decoder.
		XDM_YUV_422ILE	4	YUV 4:2:2 interleaved (little endian)
		XDM_YUV_444P	5	YUV 4:4:4 planar. Not applicable for MPEG4 decoder.
		XDM_YUV_411P	6	YUV 4:1:1 planar. Not applicable for MPEG4 decoder.
		XDM_GRAY	7	Gray format. Not applicable for MPEG4 decoder.
		XDM_RGB	8	RGB color format. Not applicable for MPEG4 decoder.

Table 4-1. List of Enumerated Data Types (continued)

Group or Enumeration Class	Bit	Symbolic Constant Name	Value	Description or Evaluation
XDM_CmdId		XDM_GETSTATUS	0	Query algorithm instance to fill Status structure.
		XDM_SETPARAMS	1	Set run time dynamic parameters via the DynamicParams structure.
		XDM_RESET	2	Reset the algorithm
		XDM_SETDEFAULT	3	Initialize all fields in Params structure to default values specified in the library.
		XDM_FLUSH	4	Handle end of stream conditions. This command forces algorithm instance to output data without additional input.
		XDM_GETBUFINFO	5	Query algorithm instance regarding the properties of input and output buffers
		XDM_GETVERSION	6	Query the algorithm's version. The result will be returned in the data field of the respective _Status structure. This control command is presently not supported.
XDM_ErrorBit	9	XDM_APPLIEDCONCEALMENT	0	Ignore
			1	Applied concealment
	10	XDM_INSUFFICIENTDATA	0	Ignore
			1	Insufficient data
	11	XDM_CORRUPTEDDATA	0	Ignore
			1	Data problem/corruption
	12	XDM_CORRUPTEDHEADER	0	Ignore
			1	Header problem/corruption
	13	XDM_UNSUPPORTEDINPUT	0	Ignore
			1	Unsupported feature/parameter in input
	14	XDM_UNSUPPORTEDPARAM	0	Ignore
			1	Unsupported input parameter or configuration
	15	XDM_FATALERROR	0	Recoverable error
			1	Fatal error (stop encoding)

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

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

The MPEG4 Decoder specific error status messages (during process call) are as follows:

- Bit 0: FAULT_HEADER_PARSING: This occurs when a fault occurs in the header decoding.
- Bit 1: FAULT_BITSTRM_DATA_PARSING: This occurs when a fault occurs in the bitstream data decoding.

4.2 Data Structures

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

Table 4-2. Data Structures

4.2.1 Common XDM Data Structures

This section includes the following common XDM data structures:

- XDM1_BufDesc
- XDM_AlgBufInfo
- IVIDEO_BufDesc
- IVIDDEC2_Params
- IVIDDEC2_DynamicParams
- IVIDDEC2_InArgs
- IVIDDEC2_Status
- IVIDDEC2_OutArgs
- IVIDDEC2_Fxns

XDM1_BufDesc

Fields

Field	Datatype	Input/Output	Description
numBufs	XDAS_Int32	Input	Number of buffers
descs[16]	XDM1_SingleBufDesc	Input	See SingleBufDesc for details.

XDM1_SingleBufDesc

Fields

Field	**Datatype**	**Input/Output**	**Description**
buf	XDAS_Int8 *	Input	Pointer to a buffer address
bufSize	XDAS_Int32	Input	Size of buf in 8-bit bytes
accessMask	XDAS_Int32	Input	Mask filled by the algorithm, declaring how the buffer was accessed by the algorithm processor. This field is not supported in this version.

XDM_AlgBufInfo

Fields

Field	Datatype	Input/Output	Description
minNumInBufs	XDAS_Int32	Input	Minimum number of input buffers
minNumOutBufs	XDAS_Int32	Input	Minimum number of output buffers
minInBufSize[16]	XDAS_Int32	Input	Minimum size required for each input buffer
minOutBufSize[16]	XDAS_Int32	Input	Minimum size required for each output buffer

Note: For MPEG4 Decoder, the buffer details are:

- Number of input buffer required is 1
- Number of output buffer required is 1 for YUV 422ILE (other formats are not supported)
- There is no restriction on input buffer size except that it should contain at least one frame of encoded data.
- The output buffer sizes (in bytes) for worst case SXVGA format are:

For YUV 422ILE:

```
Buffer = 1280 * 960 * 2
```

See the *MPEG4 Restricted Simple Profile Decoder Codec Data Sheet* (SPRS489) for more details.

IVIDEO_BufDesc

Fields

Field	Datatype	Input/Output	Description
numBufs	XDAS_Int32	Input	Number of buffers
width	XDAS_Int32	Input	Added width of a video frame
bufs[XDM_MAX_IO_BUFFERS]	XDAS_Int8 *	Input	Pointer to vector containing buffer addresses
bufSizes[XDM_MAX_IO_BUFFERS]	XDAS_Int32	Input	Size of each buffer in 8-bit bytes

IVIDDEC2_Params

Fields

Field	Datatype	Input/Output	Description
size	XDAS_Int32	Input	Size of the structure
maxHeight	XDAS_Int32	Input	Maximum height supported in pixels
maxWidth	XDAS_Int32	Input	Maximum width supported in pixels
maxBitRate	XDAS_Int32	Input	Maximum bit rate, bits per second. For example, if bit rate is 10 Mbps, set this field to 10000000. This field is not supported in this version.
dataEndianness	XDAS_Int32	Input	Endianness of output data
forceChromaFormat	XDAS_Int32	Input	Chroma format for output
maxFrameRate	XDAS_Int32	Input	Maximum frame rate in fps * 1000. For example, if max frame rate is 30 frames per second, set this field to 30000. This field is not supported in this version.

Note: Maximum video width and height supported are 1280 pixels and 960 pixels respectively.

IVIDDEC2_DynamicParams

Fields

Field	Datatype	Input/Output	Description
size	XDAS_Int32	Input	Size of the structure
decodeHeader	XDAS_Int32	Input	Decode entire access unit (0) or only header (1)
displayWidth	XDAS_Int32	Input	Pitch. If set to zero, use the decoded image width. Else, use given display width in pixels. This field is not supported in this version of MPEG4 decoder.
frameSkipMode	XDAS_Int32	Input	Video frame skip features for video decoder. This field is not supported in this version of MPEG4 decoder.
frameOrder	XDAS_Int32	Input	Video decoder output frame order. This field is not supported in this version of MPEG4 decoder.
newFrameFlag	XDAS_Int32	Input	Flag to indicate that the algorithm should start a new frame. This field is not supported in this version of MPEG4 decoder.
mbDataFlag	XDAS_Int32	Input	Flag to indicate that the algorithm should generate MB Data in addition to decoding the data. This field is not supported in this version of MPEG4 decoder.

IVIDDEC2_InArgs

Fields

Field	Datatype	Input/Output	Description
size	XDAS_Int32	Input	Size of the structure
numBytes	XDAS_Int32	Input	Size of input data in bytes, provided to the algorithm for decoding.
inputID	XDAS_Int32	Input	The decoder attaches this ID with the corresponding output frames.

Note: MPEG4 Decoder copies the inputID value to the outputID value of IVIDDEC2_OutArgs structure.

IVIDDEC2_Status

Fields

Field	Datatype	Input/Output	Description
size	XDAS_Int32	Output	Size of structure
extendedError	XDAS_Int32	Output	Extended error information
data	XDM1_SingleBufDesc	Output	Buffer descriptor for data passing. This field is not supported in this version.
maxNumDisplayBufs	XDAS_Int32	Output	The maximum number of buffers required by the codec
outputHeight	XDAS_Int32	Output	Output height in pixels
outputWidth	XDAS_Int32	Output	Output width in pixels
frameRate	XDAS_Int32	Output	Average framerate in fps*1000. This field is not supported in this version.
bitRate	XDAS_Int32	Output	Average bitrate bits per second. This field is not supported in this version.
contentType	XDAS_Int32	Output	Video content types. This field is not supported in this version.
outputChromaFormat	XDAS_Int32	Output	Output Chroma format
bufInfo	XDM_AlgBufInfo	Output	Input and output buffer information

IVIDDEC2_OutArgs

Fields

Field	Datatype	Input/Output	Description
size	XDAS_Int32	Output	Size of structure
bytesConsumed	XDAS_Int32	Output	Number of bytes consumed during the process() call
outputID[20]	XDAS_Int32	Output	Output ID corresponding to displayBufs[ ]
decodedBufs	IVIDEO1_BufDesc	Output	The decoder fills this structure with buffer pointers to the decoded frame. This field is not supported in this version.
displayBufs[20]	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. This field is not supported in this version.
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 ECD module. The pointer buffer along with the buffer size is output via this buffer descriptor. This field is not supported in this version.
freeBufID[20]	XDAS_Int32	Output	This is an array of inputID's 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. . Currently set to 0.

IVIDDEC2_Fxns

Description Defines all of the operations on IVIDDEC2 objects.

Fields

Field	Datatype	Input/Output	Description
ialg	IALG_Fxns	Output	xDAIS algorithm interface
process	XDAS_Int32*	Output	Basic video decoding call
control	XDAS_Int32*	Output	Control behavior of an algorithm

4.2.2 MPEG4 Decoder Data Structures

This section includes the following MPEG4 Decoder specific data structures:

- IMP4VDEC_Params
- Fault_inputparam_ErrorBit

IMP4VDEC_Params *Defines Creation Parameters*

Description This structure defines the creation parameters and any other implementation specific parameters for the MPEG4 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
meRange	XDAS_Int32	Input	Motion Compensation Range: 7: ME7 31: ME31 (Default) Others not supported.
displayWidth	XDAS_Int32	Input	0: Use ImageWidth as pitch (Default). Else use given Display width for pitch if displayWidth > ImageWidth
DecRotation	XDAS_Int32	Input	Rotation (anticlockwise): 0: No Rotation (Default), 1: 90 degree 2: 180 degree 3: 270 degree
UMV	XDAS_Int32	Input	UMV support: 0: OFF (Default) 1:ON

Fault_inputparam_ErrorBit *Defines Error Bit*

Description This enum structure defines the error bit for each of creation time and run time parameter for error reporting purpose.

Fields

Field Name	Bit	Values
MP4VDEC_ERROR_MAXHEIGHT	16	0: Ignore 1: Error in input height
MP4VDEC_ERROR_MAXWIDTH	17	0: Ignore 1: Error in input max width
MP4VDEC_ERROR_DATAENDIANNESS	18	0: Ignore 1: Error in data endianness
MP4VDEC_ERROR_CHROMA	19	0: Ignore 1: Error in chroma format
MP4VDEC_ERROR_ROTATION	20	0: Ignore 1: Error in rotation parameter
MP4VDEC_ERROR_DISPLAYWIDTH	21	0: Ignore 1: Error in displaywidth parameter
MP4VDEC_ERROR_MERANGE	22	0: Ignore 1: Error in ME range parameter
MP4VDEC_ERROR_DECODEHEADER	23	0: Ignore 1: Error in decode header parameter
MP4VDEC_ERROR_UMV	24	0: Ignore 1: Error in UMV parameter

4.2.3 Interface Functions

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

- *Creation* – algNumAlloc(), algAlloc(), dmaGetChannelCnt(), dmaGetChannels()
- *Initialization* – algInit(),dmaInit()
- *Control Processing* – control(), algActivate(), process(), algDeactivate()
- *Termination* – algFree()

You must call these APIs in the following sequence:

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

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* (SPRU360).

4.2.3.1 Creation APIs

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

Note: See the *MPEG4 Restricted Simple Profile Decoder Codec Data Sheet* (SPRS489) for more details on codec memory requirement.

4.2.3.2 Initialization API

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

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

```
{
      ........
      ........
    IVIDDEC2_Params    params;

    // Set the create time base parameters
    params.size = sizeof(IVIDDEC2_Params);
    params.maxHeight = 480;
    params.maxWidth = 720;
    params.maxFrameRate = XDM_DEFAULT;
    params.maxBitRate = XDM_DEFAULT;
    params.dataEndianness = XDM_BYTE;
    params. forceChromaFormat = XDM_DEFAULT;


   handle = (IALG_Handle) ALG_create((IALG_Fxns *)& MP4VDEC_TI_IMP4VDEC,
                                              (IALG_Handle) NULL,
                                              (IALG_Params *) &params)
      ........
      ........
}
```

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

```
{
      ........
      ........
    IVIDDEC2_Params            params;
    IMP4VDEC_Params            extParams;

    // Set the create time base parameters
    params.size = sizeof(IMP4VDEC_Params);
    params.maxHeight = 480;
    params.maxWidth = 720;
    params.maxFrameRate = XDM_DEFAULT;
    params.maxBitRate = XDM_DEFAULT;
    params.dataEndianness = XDM_BYTE;
    params. forceChromaFormat = XDM_DEFAULT;

    // Set the create time extended parameters

    extParams.viddecParams = params;

    extParams.meRange = 31;
    extParams.displayWidth = 720;
    extParams.DecRotation = XDM_DEFAULT;
    extParams.UMV = 1;

   handle = (IALG_Handle) ALG_create((IALG_Fxns *) & MP4VDEC_TI_IMP4VDEC,
                                           (IALG_Handle) NULL,
                                           (IALG_Params *) &extParams)
      ........
      ........
}
```

4.2.3.3 Control Processing API

The Control API is used before a call to process() to enquire about the number and size of I/O buffers or to set the dynamic params or get status of decoding. The following code gives an example for initializing and setting the base dynamic parameters for a 720x480 stream.

```
{
      ........
      ........

    IVIDDEC2_DynamicParams    dynParams;
    IVIDDEC2_Status        status;
      ........
      ........

    // Set the dynamic base parameters
    dynParams.size = sizeof(IVIDDEC2_DynamicParams);
    dynParams.decodeHeader = XDM_DEFAULT;
    dynParams.displayWidth = 720;
    dynParams.frameSkipMode = XDM_DEFAULT;
    dynParams.frameOrder = XDM_DEFAULT;
    dynParams.newFrameFlag = XDM_DEFAULT;
    dynParams.mbDataFlag = XDM_DEFAULT;

/* Set Dynamic Params */
retVal=ividDecfxns->control((IVIDDEC2_Handle)handle, XDM_SETPARAMS,
                          (IVIDDEC2_DynamicParams *)&dynamicParams,
                           (IVIDDEC2_Status *)&status);
      ........
      ........
  }
```

The current version of the MPEG4 decoder does not include any extended dynamic parameters.

4.2.3.4 Data Processing API

The Data processing API processes the input data. The following sample code gives an example of process call.

```
{
........
  retVal = ividDecfxns->process((IVIDDEC2_Handle) handle,
                                (XDM1_BufDesc *) &inputBufDesc,
                                (XDM_BufDesc *) &outputBufDesc,
                                (IVIDDEC2_InArgs *) &inArgs,
........
}
```

4.2.3.5 Termination API

The Termination terminates the algorithm instance and frees up the memory space that it uses.