MPEG4 Restricted Simple Profile Decoder on DM365

User's Guide

Literature Number: SPRUEV2A
December 2009

Preface

Read This First

About This Manual

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

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

Intended Audience

This document is intended for system engineers who want to integrate TI's codecs with other software to build a multimedia system based on the DM365 platform.

This document assumes that you are fluent in the C language, have a good working knowledge of Digital Signal Processing (DSP), digital signal processors, and DSP applications. Good knowledge of eXpressDSP Algorithm Interface Standard (XDAIS) and eXpressDSP Digital Media (XDM) standard, and IRES 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 – Revision History,** highlights the changes made to SPRUEV2 codec specific userguide to make it SPRUEV2A.

Related Documentation From Texas Instruments

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

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

Related Documentation

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

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

Abbreviations

The following abbreviations are used in this document.

Table 1-1. List of Abbreviations

Abbreviation	Description
API	Application Programming Interface
CBR	Constant Bit Rate
CCS	Code Composer Studio
CIF	Common Intermediate Format
DVD	Digital Versatile Disc
EVM	Evaluation Module
fps	Frames per second
GOB	Group of Block

Abbreviation	Description
HD	High Definition
IRES	Resource manager Specifications and Protocols
Kbps	Kilo bits per second
MJCP	MPEG JPEG Co-Processor
MPEG	Motion Picture Expert Group
NTSC	National Television Standards Committee
OBMC	Overlapped Block Motion Compensation
SXVGA	Super Extended video graphics array
VBR	Variable Bit Rate
VICP	Video and Imaging Co-Processor
VOP	Video Object Plane
XDAIS	eXpressDSP Algorithm Interface Standard
XDM	eXpressDSP Digital Media

Note:

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

Text Conventions

The following conventions are used in this document:

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

Product Support

When contacting TI for support on this codec, quote the product name (MPEG4 Restricted Simple Profile Decoder on DM365) and version number. The version number of the codec is included in the Title of the Release Notes that accompanies this codec.

Trademarks

Contents

Figures

This page is intentionally left blank

Tables

Chapter 1

Introduction

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

1.1 Overview of XDAIS, XDM, and IRES

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

1.1.1 XDAIS Overview

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

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

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

Once an algorithm instance object is created, it can 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

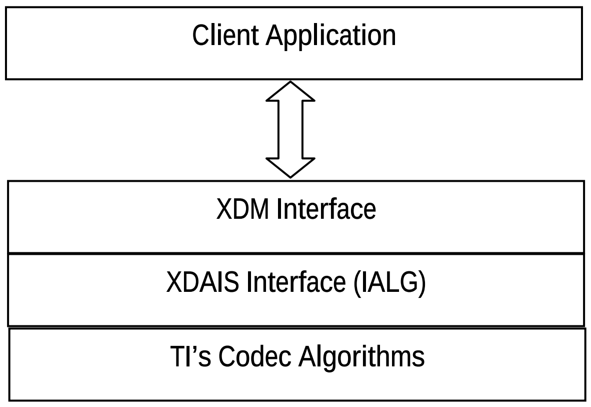
In the multimedia application space, you have the choice of integrating any codec into your multimedia system. For example, if you are building a video decoder system, you can use any of the available video decoders (such as MPEG4, H.263, or H.264) in your system. To enable easy integration with the 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 performs the basic processing (encode/decode) of data.

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

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



As 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 *XDAIS-DM (Digital Media) User Guide* (literature number SPRUEC8b).

1.1.3 IRES Overview

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

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

- **IRES -** Standard interface allowing the client application to query and provide the algorithm with its requested IRES resources.
- **RMAN -** Generic IRES-based resource manager, which manages and grants concrete IRES resources to algorithms and applications. RMAN uses a new standard interface, the IRESMAN, to support run-time registration of concrete IRES resource managers.

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

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

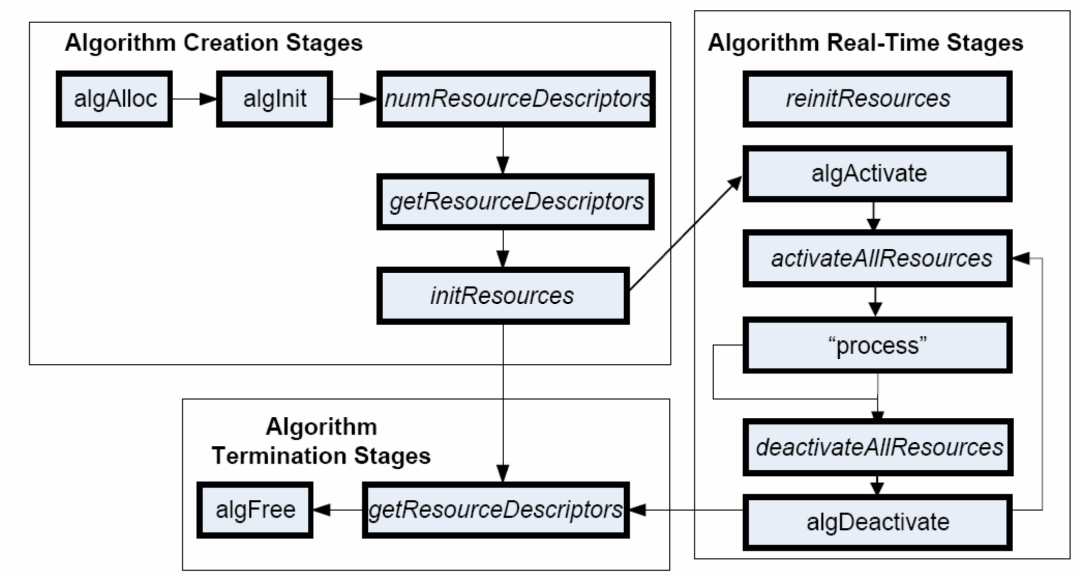


Figure 1-1. IRES Interface Definition and Function Calling Sequence.

1.2 Overview of MPEG4 Restricted Simple Profile Decoder

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

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

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

1.3 Supported Services and Features

This user guide accompanies TI's implementation of MPEG Decoder on the DM365 platform.

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

- eXpressDSP Digital Media (XDM 1.0 IVIDDEC2) interface and IRES compliant
- 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 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)
  - 1080p (1920x1080)
- 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 YUV 4:2:0 semi-planar (NV12 format, that is, Y planar Cb Cr interleaved) data as an output
- Supports Display Width feature, that is, display width can be greater than the image width

- Supports Rotation (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 DM365 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 DM365 codecs

1.4 Limitations

The following are the limitations:

- The current version of MPEG4 SP decoder is a restricted decoder. It can decode only streams encoded with the DM365 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 arbitrary width and height
  - Supports image width as multiple of 16 and height as multiple of 16.
  - Does not support image width below 160 and 192 for YUV 422ILE and YUV 420SP format respectively
- Does not support decoding of 720x1280 (rotated 720P) and 960x1280 (rotated SXVGA) formats

Chapter 2

Installation Overview

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

Topic **Page**

2.1 System Requirements for NO-OS Standalone

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

2.1.1 Hardware

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

2.1.2 Software

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

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

2.2 System Requirements for Linux

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

2.2.1 Hardware

This codec has been tested as an executable on DM365 EVM.

2.2.2 Software

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

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

2.3 Installing the Component

The codec component is released as a compressed archive. To install the codec, extract the contents of the zip file to your local hard disk. The zip file extraction creates a top-level directory called 210_V_MPEG4_D_01_10, under which another directory named mpeg4_decoder is created.

Figure 2-1 shows the sub-directories created in the mpeg4_decoder directory.

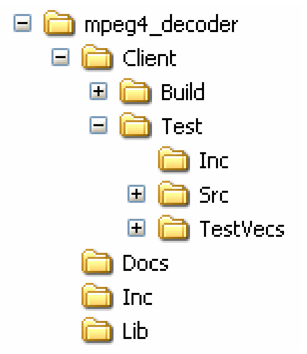


Figure 2-1.Component Directory Structure.

Table 2-1 provides a description of the sub-directories created in the mpeg4_decoder directory.

Table 2-1. Component Directories

Sub-Directory	Description
mpeg4_decoder/Client/Build	This folder is available only in NO-OS Standalone release package. Not required for Linux release package. Contains make file, cmd file and configuration file to build the NO-OS standalone test application.
mpeg4_decoder/Client/Test/Inc	Contains header files needed for the application code
mpeg4_decoder/Client/Test/Src	Contains application C files, makefile, and configuration file. Executable will be built in this folder.
mpeg4_decoder/Client/Test/TestVecs	Contains test vectors, configuration files
mpeg4_decoder/Docs	Contains user guide, datasheet, and release notes
mpeg4_decoder/Inc	Contains MPEG4 Decoder Interface file
mpeg4_decoder/Lib	Contains MPEG4 Decoder and other support libraries

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

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

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

2) Verify if the codec object library mp4vdecAlg.lib exists in the \Lib sub-directory.

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

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

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

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

7) Type the command “**gmake –f mp4vdecTestApp.mak**” at the prompt and this generates an executable file, mp4vdecApp.out in the \Client\Build\Out sub-directory.

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

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

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

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

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

4) Open Setup Code Composer Studio version 3.3.

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

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

7) Right click on ARM926 and connect.

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

9) Add the GEL file and initialize it properly

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

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

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

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

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

12) Halt the coprocessor from Code Composer Studio IDE.

2.6 Building and Running the Sample Test Application on Linux

The sample test application that accompanies this codec component will take 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 you have installed Framework Component (FC), XDC, and LSP. For information about the version, see Release Note.

2) Verify that libmp4vdec.a library is present in mpeg4_decoder/Lib directory.

3) Change directory to mpeg4_decoder/Client/Test/Src and type `make clean` followed by a `make` command. This will use the Makefile in that directory to build the test executable mp4dec-r into the mpeg4_decoder/Client/Test/Src directory.

Note:

ARM tool chain, that is, arm_v5t_le-gcc (ARM gcc) compiler path needs to be set in user's environment path before building the MPEG4 decoder executable.

4) To run the mp4dec-r executable on the DM365 EVM board:

   a) Set up the DM365 EVM Board.

      For information about setting up the DM365 environment, see the DM365 Getting Started Guide available in doc directory in DVSDK release package.

   b) Run the MPEG4 Decoder executable:

      i) Ensure that complete Client folder is in target file system..

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

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

      iv) Load the kernel modules by executing folliwng command.
```
$./loadmodules.sh
```

      Change the directory to Client/Test/Src folder and execute following command to run the JPEG decoder executable.

```
$./mp4dec-r
```

      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-r –ext`

2.7 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 earlier.

2.7.1 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 2.7.
- ❑ `Input` is the input file name (use complete path).
- ❑ `Output` is the output file name.

A sample Testvecs.cfg file is as shown.

```
0
../TestVecs/Config/Testparams.cfg
../TestVecs/Input/colorful_toys_cif_5frms.bits
../TestVecs/Output/colorful_toys_cif_422ile.yuv
```

2.7.2 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    = 1

forceChromaFormat = 4     # 4:422ILE, 9: 420 SemiPlanar
```

2.7.3 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    = 1
forceChromaFormat = 4     # 4: 422ILE, 9: 420 SemiPlanar
rotation       = 0    #
                          0,90,180,270 degrees
displayWidth      = 0    #Display width of application buffer

meRange           = 31   #only 7 and 31 are supported

unrestrictedMV               = 1    #UMV support
```

This page is intentionally left blank

Chapter 3

Sample Usage

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

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:

1) 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**
31 channels(channel 63 compulsory)	All paramsets associated with TCCS and 16 additional paramsets.

In addition to these 31 TCCs requirements, it also needs 16 more PaRamSets that are allocated through IRES interface.

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

3) 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.

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

5) If 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 the required timing requirements between video/image frames.

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

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

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

3.2 Overview of the Test Application

The test application exercises the `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.

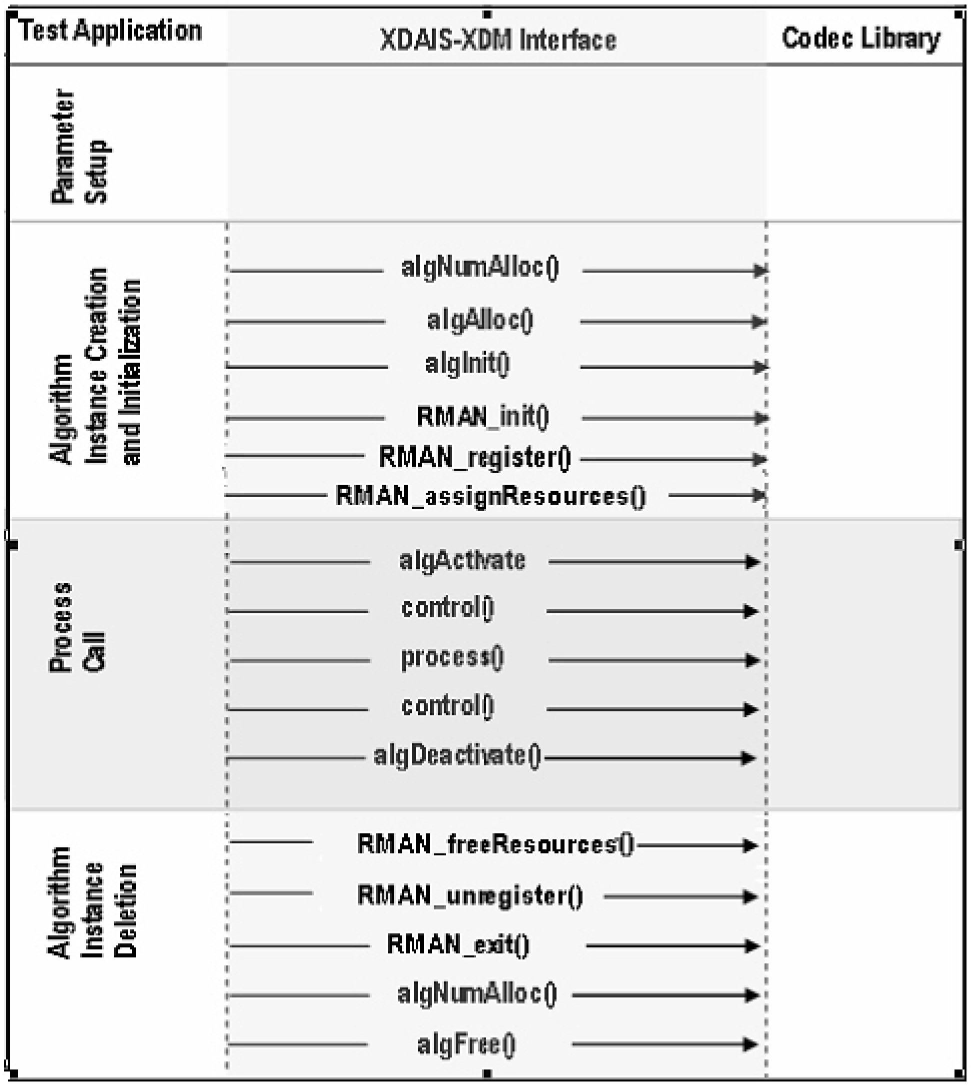


Figure 3-1. Test Application Sample Implementation.

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, 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) 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:

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

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

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

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

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

3.2.3 Process Call in Single Instance Scenario

After algorithm instance creation and initialization, the test application 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 this section and the constraints as described in section 3.2.1 are applicable.

Note:

`algActivate ()` is a mandatory call before 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 up for the application.

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

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

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

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

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

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.

1) `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.

2) `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.

3.2.5.1 Process Call with algActivate and algDeactivate

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

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

2) Calls `algActivate()`, which initializes the 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 3.2.1 are applicable.

3.3 Frame Buffer Management by Application

3.3.1 Frame Buffer Input and Output

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

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 1st frame, if the above control call returns a smaller size than that is returned in the first call

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

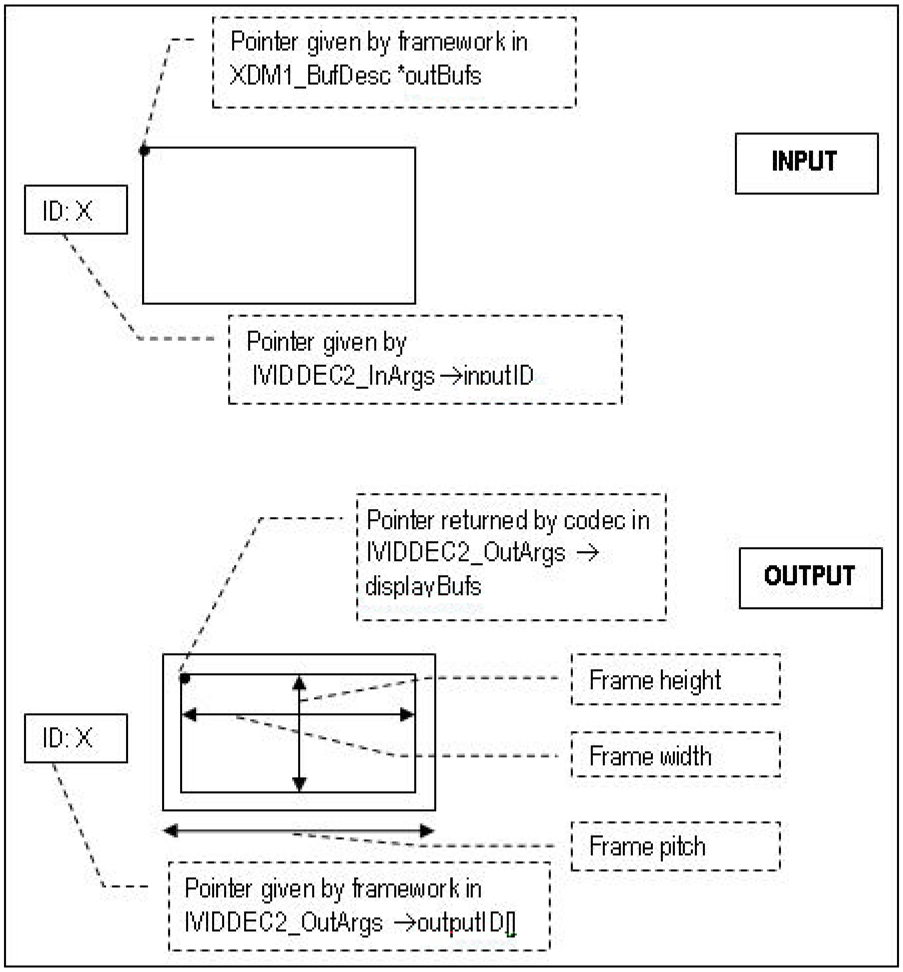

Pointer given by framework in XDM1_BufDesc *outBufs
INPUT
ID: X
Pointer given by IVIDDEC2_InArgs →inputID
Pointer returned by codec in IVIDDEC2_OutArgs → displayBufs
OUTPUT
Frame height
ID: X
Frame width
Frame pitch
Pointer given by framework in IVIDDEC2_OutArgs →outputID[]

Figure 3-2. Frame Buffer Pointer Implementation.

Note:

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

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

Note:

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

3.3.2 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.

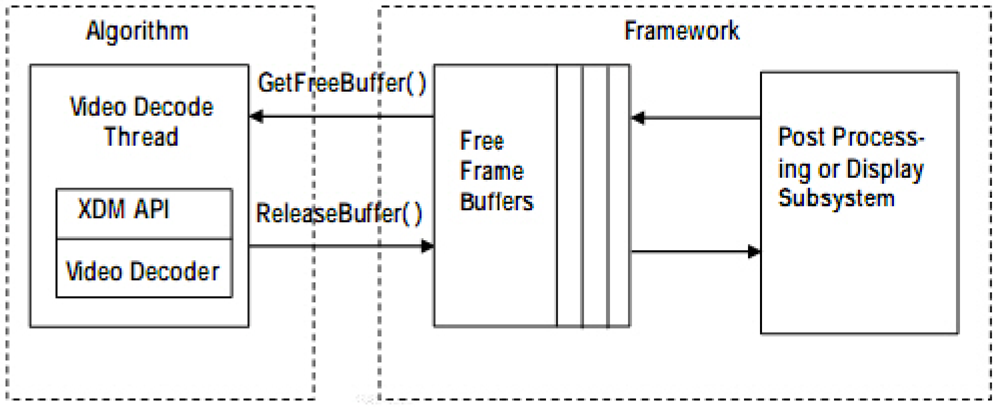


Figure 3-3. Interaction of Frame Buffers Between Application and Framework.

The sample application also provides a prototype for managing frame buffers. It implements the following functions, which are defined in file buffermanager.c provided along with test application:

- `BUFFMGR_Init() - BUFFMGR_Init` function is called by the test application to initialize the global buffer element array to default and to allocate one frame of memory data for the first process call output buffers depending on the maximum width and maximum height supplied in params.
- `BUFFMGR_ReInit() - BUFFMGR_ReInit` function allocates the remaining luma and chroma buffers required by the stream based on the actual picture width and height obtained after first process call.
- `BUFFMGR_GetFreeBuffer() - BUFFMGR_GetFreeBuffer` function searches for a free buffer in global buffer array and returns the address of that element. Incase, if no elements are free, then it returns NULL.
- `BUFFMGR_ReleaseBuffer() - BUFFMGR_ReleaseBuffer` function takes an array of buffer-IDs, which are released by the test-app. Zero is not a valid buffer ID. Hence, this function keeps moving until it encounters a buffer ID as zero or it hits the `MAX_BUFF_ELEMENTS`.
- `BUFFMGR_DeInit()- BUFFMGR_DeInit` function releases all memory allocated by buffer manager.

This page is intentionally left blank

Chapter 4

API Reference

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

Topic **Page**

4.1 Symbolic Constants and Enumerated Data Types

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

Table 4-1. List of Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	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.

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
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.
	XDM_YUV_420SP	9	YUV 4:2:0 semi planar format
XDM_CmdId	XDM_GETSTATUS	0	Query algorithm instance to fill `Status` structure.
	XDM_SETPARAMS	1	Set run-time dynamic parameters through the `DynamicParams` structure.
	XDM_RESET	2	Reset the algorithm
	XDM_SETDEFAULT	3	Initialize all fields in `Params` structure to default values specified in the library.
	XDM_FLUSH	4	Handle end of stream conditions. This command forces algorithm instance to output data without additional input.
	XDM_GETBUFINFO	5	Query algorithm instance regarding the properties of input and output buffers
	XDM_GETVERSION	6	Query the algorithm's version. The result will be returned in the data field of the respective _Status structure. This control command is currently not supported.

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
`XDM_ErrorBit`	`XDM_APPLIEDCONCEALMENT`	9	Bit 9 ❑ 1 - Applied concealment ❑ 0 - Ignore
	`XDM_INSUFFICIENTDATA`	10	Bit 10 ❑ 1 - Insufficient data ❑ 0 - Ignore
	`XDM_CORRUPTEDDATA`	11	Bit 11 ❑ 1 - Data problem/corruption ❑ 0 - Ignore
	`XDM_CORRUPTEDHEADER`	12	Bit 12 ❑ 1 - Header problem/corruption ❑ 0 - Ignore
	`XDM_UNSUPPORTEDINPUT`	13	Bit 13 ❑ 1-Unsupported feature/parameter in input ❑ 0 - Ignore
	`XDM_UNSUPPORTEDPARAM`	14	Bit 14 ❑ 1- Unsupported input parameter or configuration ❑ 0 - Ignore
	`XDM_FATALERROR`	15	Bit 15 ❑ 1 - Fatal error (stop encoding) ❑ 0 - Recoverable error

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:

1) `Bit 0: FAULT_HEADER_PARSING`: This occurs when a fault occurs in the header decoding

2) `Bit 1: FAULT_BITSTRM_DATA_PARSING`: This occurs when a fault occurs in the bit-stream 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.

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`

4.2.1.1 XDM1_BufDesc

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
`numBufs`	`XDAS_Int32`	Input	Number of buffers
`descs[16]`	`XDM1_SingleBufDesc`	Input	See `XDM1_SingleBufDesc` for details

4.2.1.2 XDM1_SingleBufDesc

‖ Description

This structure contains elements required to hold one data buffer.

‖ Fields

Field	Data Type	Input/ Output	Description
`buf`	`XDAS_Int8 *`	Input	Pointer to a buffer address
`bufSize`	`XDAS_Int32`	Input	Size of `buf` in 8-bit bytes.

Field	Data Type	Input/ Output	Description
`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.

4.2.1.3 XDM1_AlgBufInfo

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
`minNumInBufs`	`XDAS_Int32`	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 & 2 for YUV 420SP(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 Decoder Data Sheet* for more details.

4.2.1.4 IVIDEO1_BufDesc

|| Description

This structure defines the buffer descriptor for input video buffers.

|| Fields

Field	Data Type	Input/ Output	Description
`numBufs`	`XDAS_Int32`	Input	Number of buffers.
`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.

4.2.1.5 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	Data Type	Input/ Output	Description
`size`	`XDAS_Int32`	Input	Size of the structure
`maxHeight`	`XDAS_Int32`	Input	Maximum height supported in pixels. Valid Range: 16 to 1920 Default: 960
`maxWidth`	`XDAS_Int32`	Input	Maximum width supported in pixels Valid Range: 160 to 1920 Default : 1280
`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. Only `XDM_BYTE` (1) is supported.
`forceChromaFormat`	`XDAS_Int32`	Input	Chroma format for output. Only 4: YUV422ILE (default) and 9: YUV420 SemiPlanar is supported

Field	Data Type	Input/ Output	Description
`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 1920 pixels and 1920 pixels respectively.

4.2.1.6 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	Data Type	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 this is set to zero, uses the decoded image width. Otherwise, uses the given display width in pixels. This field is not supported in this version of MPEG4 decoder. Display width feature is supported using extended create time parameter. See `IMP4VDEC_PARAMS` structure for details.
`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.

Field	Data Type	Input/ Output	Description
`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.

4.2.1.7 *IVIDDEC2_InArgs*

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
`size`	`XDAS_Int32`	Input	Size of the structure
`numBytes`	`XDAS_Int32`	Input	Size of input data in bytes, provided to the algorithm for decoding.
`inputID`	`XDAS_Int32`	Input	The decoder will attach this ID with the corresponding output frames. Should be non-zero value.

Note:

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

4.2.1.8 *IVIDDEC2_Status*

‖ Description

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

‖ Fields

Field	Data Type	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.

Field	Data Type	Input/ Output	Description
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 bit-rate bits per second. This field is not supported in this version.
contentType	XDAS_Int32	Output	Video content types.
outputChromaFormat	XDAS_Int32	Output	Output Chroma format.
bufInfo	XDM_AlgBufInfo	Output	Input and output buffer information.

4.2.1.9 IVIDDEC2_OutArgs

|| Description

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

|| Fields

Field	Data Type	Input/ Output	Description
size	XDAS_Int32	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 through this buffer descriptor. This field is not supported in this version.

Field	Data Type	Input/ Output	Description
`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.

4.2.1.10 IVIDDEC2_Fxns

‖ Description

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

‖ Fields

Field	Data Type	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`

4.2.2.1 IMP4VDEC_Params

‖ Description

This structure defines the creation parameters and any other implementation specific parameters (extended parameters) for the MPEG4 Decoder instance object. The creation parameters are defined in the XDM data structure, `IVIDDEC2_Params`.

‖ Fields

Field	Data Type	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 are not supported. Note: If `meRange` is set to 31, streams encoded with `meRange` 7 and 31 can be decoded. If `meRange` is set to 7, only stream encoded with `meRange` 7 can be decoded (else output will have artifacts).
`displayWidth`	`XDAS_Int32`	Input	0: Use `ImageWidth` as pitch (Default). Otherwise, use given `displayWidth` for pitch if `displayWidth > ImageWidth`
`rotation`	`XDAS_Int32`	Input	Rotation (anticlockwise): ❑ 0: No Rotation (Default) ❑ 90: 90 degree ❑ 180: 180 degree ❑ 270: 270 degree
`unrestrictedMV`	`XDAS_Int32`	Input	UMV support : ❑ 0: OFF (Default) ❑ 1:ON Note: If input stream has unrestricted MV, this parameter must be set to ‘1’ for correct decoding.

4.2.2.2 Fault_inputparam_ErrorBit

‖ Description

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

‖ Fields

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

4.3 Interface Functions

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

- **Creation** – `algNumAlloc(), algAlloc(), 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).

4.3.1 Creation APIs

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

Note:

See the *MPEG4 Decoder Data Sheet* for more details on codec memory requirement.

4.3.2 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.rotation = 0;
    extParams.unrestrictedMV = 1;

handle = (IALG Handle) ALG create((IALG Fxns *)&
MP4VDEC TI IMP4VDEC, (IALG Handle) NULL, (IALG Params *)
&extParams)
      .........................
      .........................

}
```

4.3.3 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 is 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);
      ……………………
      ……………………
  }
```

4.3.4 Data Processing API

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

```
{

     ……………………
      ……………………

          retVal = ividDecfxns-
>process((IVIDDEC2 Handle) handle,(XDM1 BufDesc *)
&inputBufDesc,(XDM1 BufDesc *) &outputBufDesc,
(IVIDDEC2 InArgs *) &inArgs,
      ……………………
      ……………………

}
```

4.3.5 Termination API

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

Chapter 5

Revision History

This revision history highlights the changes made to SPRUEV2 codec specific user guide to make it SPRUEV2A..

Table 5-1. MPEG4 Restricted Simple Profile Decoder on DM365

Section	**Addition/Deletion/Modification**
Section 1.3	Supported Services and Features: ❑ Added Support for 1080p decoding
Section 2.1	System Requirements for NO-OS Standalone: ❑ Added Hardware and software requirements for NO-OS Standalone
Section 2.2	System Requirements for Linux : ❑ Added Hardware and software requirements for Linux
Section 2.3	Installing the Component: ❑ Modified top-level directory name
Section 2.4	Building the Sample Test Application for EVM Standalone (NO-OS): ❑ Added procedure to Build and Run the Sample Test Application
Section 2.6	Building and Running the Sample Test Application on Linux: ❑ Added procedure to Build and Run the Sample Test Application on Linux
Section 2.7.1	Test Configuration files : ❑ Modified sample Testvecs.cfg file
Section 2.7.3	Decoder Configuration File for Extended Parameters: ❑ Modified decoder configuration file, Testparams.cfg
Section 4.2.1.5	`IVIDDEC2_Params:` Modified valid range of values for the following fields ❑ `maxHeight` ❑ `maxWidth`