Motion Compensated Temporal Noise Filter on DM388 Platform

User Guide

January 2014

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

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

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

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

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

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

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

1 Read This First

1.1 About This Manual

This document describes how to install and work with Texas Instruments' (TI) Motion Compensated Temporal Noise Filter (MCTNF) implemented on the DM388 platform. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

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

1.2 Intended Audience

This document is intended for system engineers who want to integrate TI's algorithms with other software to build a multimedia system based on the DM388 and Visual C.

This document assumes that you are fluent in the C language, and aware of video processing applications. Good knowledge of eXpressDSP Algorithm Interface Standard (XDAIS) and eXpressDSP Digital Media (XDM) standard will be helpful.

1.3 How to Use This Manual

This document includes the following chapters:

- **Chapter 2 - Introduction**, provides a brief introduction to the XDAIS and XDM standards. It also provides an overview of the MCTNF and lists its supported features.
- **Chapter 3 - Installation Overview**, describes how to install, build, and run the algorithm.
- **Chapter 4 - Sample Usage**, describes the sample usage of the algorithm.
- **Chapter 5 - API Reference**, describes the data structures and interface functions used in the algorithm.
- **Chapter 6 - Frequently Asked Questions,** provides answers to frequently asked questions related to using this video noise filter.

1.4 Related Documentation From Texas Instruments

This document frequently refers TI's DSP algorithm standards such as, XDAIS and XDM. To obtain a copy of document related to any of these standards, visit the Texas Instruments website at www.ti.com. One of those referred document is listed below

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

1.5 Abbreviations

The following abbreviations are used in this document.

Table 1 List of Abbreviations

Abbreviation	Description
API	Application Programming Interface
CIF	Common Intermediate Format
DMA	Direct Memory Access
DMAN3	DMA Manager
DSP	Digital Signal Processing
EVM	Evaluation Module
IRES	Interface for Resources
MCTNF	Motion Compensated Temporal Noise Filter
QCIF	Quarter Common Intermediate Format
QVGA	Quarter Video Graphics Array
RMAN	Resource Manager
SQCIF	Sub Quarter Common Intermediate Format
VGA	Video Graphics Array
XDAIS	eXpressDSP Algorithm Interface Standard
XDM	eXpressDSP Digital Media

1.6 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.

1.7 Product Support

When contacting TI for support on this product, quote the product name (MCTNF on DM388 platform) and version number. The version number of the MCTNF is included in the Title of the Release Notes that accompanies the product release.

1.8 Trademarks

Code Composer Studio, eXpressDSP, DM388 are trademarks of Texas Instruments.

2 Introduction

This chapter provides a brief introduction to XDAIS and XDM. It also provides an overview of TI's implementation of the MCTNF on the DM388 platform and its supported features.

2.1 Overview of XDAIS, XDM

TI's multimedia algorithms implementations are based on the eXpressDSP Digital Media (XDM) standard. XDM is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS). Please refer documents related to XDAIS for further details.

2.1.1 XDAIS Overview

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

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

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

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

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

2.1.2 XDM Overview

In the multimedia application space, you have the choice of integrating any MCTNF 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 MCTNFs with similar functionality use similar APIs. XDM was primarily defined as an extension to XDAIS to ensure uniformity across different classes of MCTNFs (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 MCTNFs, XDM also standardizes the generic parameters that the client application must pass to these APIs. The client application can define additional implementation specific parameters using extended data structures.

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

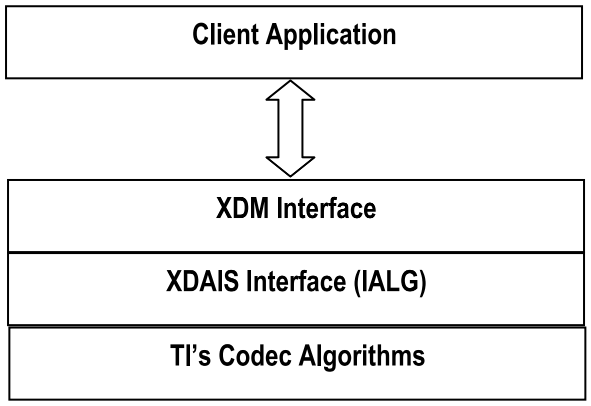

Figure 1 XDM Introduction

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

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

2.2 Overview of MCTNF

Motion Compensated Temporal Noise Filter is designed to remove video noise effectively without distorting the video signal. It compensates motion between the frames first before performing a weighted average filtering. To compensate motion between frames, full fledged motion estimation is performed at block level.

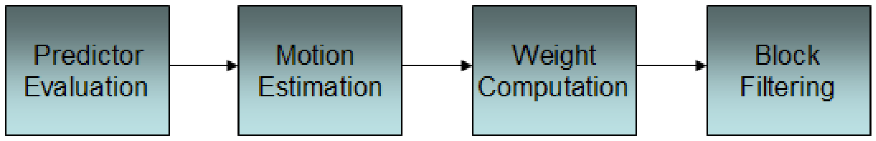

Figure 2 Fundamental blocks of MCTNF

The basic building blocks of MCTNF are Predictor evaluation, Motion estimation, Weight computation and Block filtering as shown in the figure. The final filtered frame will be used as reference to the subsequent frames.

2.3 Supported Services and Features

This user guide accompanies TI's implementation of MCTNF Algorithm on the DM388 platform.

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

- Supports 4:2:0 chroma format.
- Supports only progressive sequence.
- Supports 8 bit input pixel precision.
- Supports one reference frame.
- Supports arbitrary video resolutions from 96x80 upto 2048x2048, along with only restriction of width being multiple of 16, and height being multiple of 2.
- Support for user control to tune the behavior of filtering.
- Support of key parameter (like average Sad, global motion vector,filter strength etc. Refer sectionIMCTNF_OutArgs) output to user to do extra processing or decision post MCTNF.
- Independent of any operating system.

This version of the MCTNF does not support following features:

- MCTNF does not operate in BASE CLASS only mode.
- Multiple reference frame for motion estimation and compensation.

3 Installation Overview

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

3.1 System Requirements

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

3.1.1 Hardware

This algorithm has been built and tested on DM388 platform.

3.1.2 Software

The following are the software requirements for the stand alone functioning of the MCTNF:

- **Development Environment:** This project is developed using Ti's Code Generation Tool 5.0.5. Other required tools used in development are mentioned in section 3.3
- The project are built using g-make (GNU Make version 3.78.1). GNU tools comes along with CCS installation.

3.2 Installing the Component

The algorithm component is released as compressed archive. Following sub sections provided details on installation along with directory structure.

3.2.1 Installing the compressed archive

The algorithm component is released as a compressed archive. To install the algorithm, extract the contents of the zip file onto your local hard disk. The zip file extraction creates a top-level directory called 100.V.MCTNF.DM388.00.00, under which directory named DM388_001 is created. Folder structure of this top level directory is shown in below figure.

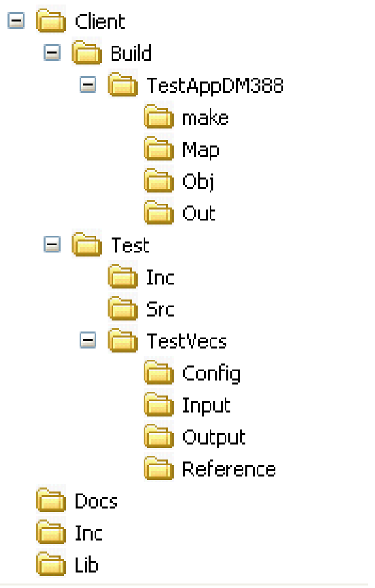

Figure 3 Component Directory Structure In case of Object Release

Table 2 Component Directories in case of Object release

Sub-Directory	Description
\Client\Build\TestAppDM388\make	Contains GNU make file for building client test application
\Client\Build\TestAppDM388\Map	Place for map file generated after building the test application
\Client\Build\TestAppDM388\Obj	Place for intermediate Object files generated after building host test application
\Client\Build\TestAppDM388\Out	Place for final application executable (.out) file generated by the sample test application building.
\Client\Test\Inc	Contains standalone test application header files
\Client\Test\Src\	Contains standalone test application source files
\Client\Test\TestVecs\Config	Contains sample configuration files for MCTNF algorithm
\Client\Test\TestVecs\Input	Contains input test vectors
\Client\Test\TestVecs\Output	Contains output generated by the algorithm.
\Client\Test\TestVecs\Reference	Contains read-only reference output to be used for cross-checking against algorithm output in the folder \Client\Test\TestVecs\Output

Sub-Directory	Description
\docs	Contains user guide, data sheet and release notes for specific release
\Inc	Contains XDM related header files, which allow interface to the algorithm library.
\Lib	Contains the library file named as mctnf_ti_host.lib

3.3 Building Sample Test Application

This MCTNF algorithm library has been accompanied by a sample test application. To run the sample test application Framework Components, XDC tools and HDVICP2 library are required.

This version of the MCTNF algorithm library has been validated with XDAIS tools containing IVIDNF1 interface version. Other required components (for test application building) version details are provided below.

The version of the XDAIS required is 7.10

The version of the XDC tools required is 3.22.04.46

The version of the Code Generation tools required is 5.0.5

The version of Framework Components required is 3.21.3.34

Make sure that system environment variable “CG_TOOL_ROOT” is set to appropriate code generation tool installation path.

3.3.1 Installing XDAIS tools(XDAIS)

XDAIS version 7.10 can be downloaded from the following website:

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/xdais/7_21_01_07/index_FDS.html

Extract the XDAIS zip file to the same location where Code Composer Studio has been installed. For example:

C:\CCStudio5.0

Set a system environment variable named XDAIS_INSTALL_DIR pointing to <install directory>\<Xdais_directory>

3.3.2 Installing XDC Tools

XDC Tools are required to build the test application. The test application uses the standard files like <std.h> from XDC tools. The xdc tools can be downloaded and installed from the following website:

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/rtsc/3_22_04_46/index_FDS.html

Also Ensure that the environment variable XDCROOT is set to the XDC installation directory. Eg:set XDCROOT to <install directory>\<xdc_tools>

Set a system environment variable named XDC_INSTALL_DIR pointing to <install directory>\<xdc_directory>

3.3.3 Installing Framework Component (FC)

Framework Components are required for using IRES interface for EDMA3 hardware.FC tools can be downloaded and installed from the following website

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/fc/3_21_03_34/index_FDS.html

Mare sure that FC tools are recognized as RTSC package in CCS V5 by adding installation directory path RTSC products search path in CCS V5 preferences.

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

3.3.4 Installing Code Generation Tools

Install Code generation Tools version 5.0.5 from the link

http://syntaxerror.dal.design.ti.com/release/releases/arm/rel5_0_5/build/install/

After installing the CG tools, set the environment variable to CG_TOOL_DIR to the installed directory like <install directory>\<cgtools_directory>

3.3.5 Building the Test Application Executable through GMAKE

The sample test application that accompanies this MCTNF component will run in TI's Code Composer Studio development environment. To build and run the sample test application through gmake, follow these steps:

1) Verify that you have installed code generation tools version 5.0.5.

2) Verify that appropriate environment variables has been set as discussed in this above sections.

3) Verify that the following MCTNF object libraries exist in \Lib sub-directory

4) Build the sample test application project by gmake

a) Client\Build\TestAppDeviceName\make> gmake -s deps

b) Client\Build\TestAppDeviceName\make> gmake -k -s all

5) All files required for this project are available at the path \Client\Build\TestAppDeviceName

7) The above step creates an executable file, mctnf_ti_host.lib.out in the \Client\Build\TestAppDeviceName\Out sub-directory.

8) Open CCS with DM388 platform selected configuration file. Select Target > Load Program on M3_Video, browse to the \Client\Build\ TestAppDeviceName\Out sub-directory, select the MCTNF executable created in step 6, and load it into Code Composer Studio in preparation for execution.

9) Select Target > Run on M3_Video window to execute the sample test application.

10) sample test application takes the input files stored in the \Client\Test\TestVecs\Input sub-directory, runs the MCTNF. The reference files stored in the \Client\Test\TestVecs\Reference sub-directory can be used to verify that the filtering is functioning as expected.

11) On failure, the application exits with the message “Frame processing failed”.

12) On successful completion, the application displays the information for each frame and generates output 264 bit-stream in \Client\Test\TestVecs\Output directory. User should compare with the reference provided in \Client\Test\TestVecs\Reference directory. Both the YUV content should be same to conclude successful execution

3.4 Configuration File

This algorithm is shipped along with:

- ❑ Algorithm configuration file (mctnf.cfg) – specifies the configuration parameters used by the test application to configure the Algorithm.
- ❑ TestCases.txt – This file has list of config files, these needs to be executed with parameter (integer) preceding. The meaning of the parameter is below.
  - ➢ 0 – execute the test case
  - ➢ 1 – Skip the test case.
  - ➢ 2 – Terminate the regression

3.4.1 Test Application Configuration File

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

A sample mctnf.cfg file is as shown.

```
# New Input File Format is as follows
# ParameterName = ParameterValue # Comment
#########################################################################################
# Files
#########################################################################################
InputFile    = "..\..\..\Test\TestVecs\Input\fruits_p176x144_nv12.yuv"
OutputFile   = "..\..\..\Test\TestVecs\Output\fruits_p176x144.yuv"

#########################################################################################
# MCTNF Control
#########################################################################################
FilterPreset             = 0        # Only supported value is 0.
NumInputUnits            = 10       # Total number of frames to filter
MaxWidth                 = 1920     # Max Frame width should be multiple of 16
MaxHeight                = 1088     # Max Frame height should be multiple of 16
InputChromaFormat        = 9        # Input data chroma format, supports only DM_YUV_420SP
format
InputContentType         = 0        # Input buffer content type, 0 -> Progressive Type, 1->
Interlaced. Only supported value is 0
InputDataMode            = 3        # Process entire frame. Only supported value is 3.
OutputDataMode           = 3        # Process entire frame. Only supported value is 3.
NumOutputUnits           = 1        # Number of units of output-data. Only supported value is
1.
dataLayout               = 0        # input data buffer layout 0=> interleaved, 1=> seprated.
Applicable only for interlace content.
inputWidth               = 176      # width of image
inputHeight              = 144      # Height of image
numPastRef               = 1        # Number of Past Reference Frames. Only supported value is
1
numFutureRef             = 0        # Number of Future Reference Frames. Only supported value
is 0
captureWidth             = 176      # Image capture width
captureHeight            = 144      # Image capture height
captureTopLeftx          = 0        # Exact source position of the pixel to encode in input
buffer X direction
captureTopLefty          = 0        # Exact source position of the pixel to encode in input
buffer Y direction

##########################################################################################
# Motion Search Control
##########################################################################################
motionSearchPreset       = 1        # Inter coding mode preset, 0 => deafult values, 1 => user
defined
searchRangeHor           = 144      # Horizontal Search Range for P frames in integer pixels,
e.g. 144 -> this will make search range from (-144 to +144) offseted by GMV.x
searchRangeVer           = 32       # Vertical Search Range for P frames  in integer pixels
searchCenter_x           = 32767    # Search center for Motion estimation i.e global motion
vector in X dir. For best video quality, set this to 32767 so that MCTNF will set
searchCenter_x and searchCenter_y at internally computed global offset.
searchCenter_y           = 0        # Search center for Motion estimation i.e global motion
vector in Y dir
```

```
##########################################################################################
# Noise Filter Control
##########################################################################################
meEffectivenessTh        = 2147483647 # if avg mv cost is higher than this condider ME is not
effective. Max Value (INT_MAX(32) = 2^31-1) = 2147483647
meLambdaType             = 0       # adaptive Lambda usage for ME search. 0:fixed
(default),1:pic level, based on log(sad) ,2:block level,3:pic level based on SAD, linear
relation
meLambdaFactorQ2         = 96      # lambda for ME, default: 24
maxLambdaQ2              = 255     # max lambda in case of dynamic lambda, default:72
sadForMinLambda          = 200     # min lambda upto this SAD, default:200
biasZeroMotion           = 1       # search explicitly at <0,0> 0: disable (default), 1:fixed
th, 2:th=lambda/2 3: Lambda absorption inside mv cost for explicit zero check
blendingFactorQ4         = 0       # blend factor, 0:dynamic, non zero:fixed
minBlendQ4               = 24      # min blend, value 0 indicates internally chosen
maxBlendQ4               = 40      # max blend,value 0 indicates internally chosen
fixWtForCurQ8            = 0       # fixed current wt for each MB, 0:dynamic (default)
minWtForCurQ8            = 41      # min wt for current pixels for avoiding excesive filtering,
default (0)
staticMBThZeroMV         = 10      # if cur MB is Zero MV and num of Zero MV neighbor >= th,
then call cur Mb static
staticMBThNonZeroMV      = 10      # If Cur MB is Non Zero and num of Zero MV neighbor >= th,
then call cur Mb static
blockinessRemFactor      = 0       # deblock filter strength 5 indicates dynamic filter, 0-3
indicates filter strength, 0 no filter, 3 max filter.
sadForMaxStrength        = 250     # SAD value above which max strength will be used
mvXThForStaticLamda      = 5       # Threshold in Q2 format for MVx to determine block level
lambda. If local motion in x direction is above this threshold then smaller lambda will be
used.
mvYThForStaticLamda      = 5       # Threshold in Q2 format for MVy to determine block level
lambda. . If local motion in y direction is above this threshold then smaller lambda will be
used.
```

Any field in the `IVIDNF1_Params` structure (see Section 5.2.1.5) can be set in the mctnf.cfg file using the syntax as shown in the code snippet. If you specify additional fields in the mctnf.cfg file, ensure that you modify the test application appropriately to handle these fields.

3.5 Standards Conformance and User-Defined Inputs

To check the conformance of the algorithm for the default input file shipped along with the algorithm, follow the steps as described in Section **Error! Reference source not found.**.

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

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

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

3) Edit the configuration file, testparams.cfg available in the \Client\Test\TestVecs\Config sub-directory. For details on the format of the mctnf.cfg file, see Section 3.4.1.

3.6 Uninstalling the Component

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

4 Sample Usage

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

4.1 Overview of the Test Application

The test application exercises the `IVIDNF1` and extended class of the MCTNF library. The source files for this application are available in the \Client\Test\Src and \Client\Test\Inc sub-directories.

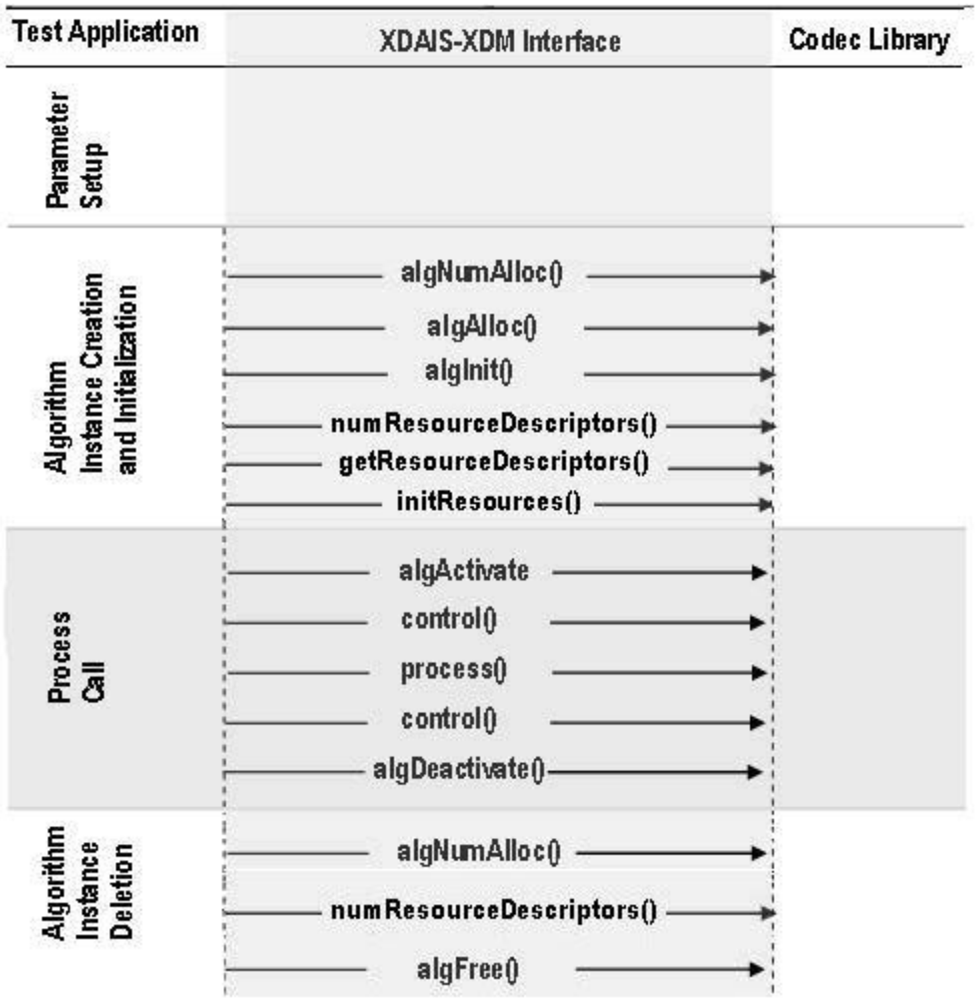

Table 3 Test Application Sample Implementation

The test application is divided into four logical blocks:

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

4.2 Parameter Setup

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

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

1) Opens the configuration file, listed in TesCases.txt and reads the various configuration parameters required for the algorithm.

For more details on the configuration files, see Section 3.4.

2) Sets the `interface` structure based on the values it reads from the configuration file.
3) Does the algorithm instance creation and other handshake via. control methods
4) For each frame reads the input yuv frame into the application input buffer and makes a process call
5) For each frame dumps out the generated/filtered YUV frame into the specified file

4.3 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 implemented by the algorithm are called in sequence by `ALG_create()`:

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

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

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

1) `numResourceDescriptors()` - To understand the number of resources needed by algorithm.

2) `getResourceDescriptors()` – To get the attributes of the resources.

3) `initResources()` - After resources are created, application gives the resources to algorithm through this API

4.4 Process Call

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

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

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

3) Calls the `process()` function to filter a single frame of data. The behavior of the algorithm can be controlled using various dynamic parameters (see Section 5.2.1.6). The inputs to the process function are input and output buffer descriptors, pointer to the `IVIDNF1_InArgs` and `IVIDNF1_OutArgs` structures.

4) When the `process()` function is called for filtering a single frame of data, the software triggers the start of algorithm.

The `control()` and `process()` functions should be called only within the scope of the `algActivate()` and `algDeactivate()` XDAIS functions, which activate and deactivate the algorithm instance respectively. If the same algorithm is in-use between two process/control function calls, calling these functions can be avoided. Once an algorithm is activated, there can be any ordering of `control()` and `process()` functions. The following APIs are called in sequence:

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

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

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

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

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

The do-while loop encapsulates frame level `process()` call and updates the input buffer pointer every time before the next call. The do-while loop breaks off either when an error condition occurs or when the input buffer exhausts.

If the algorithm uses any resources through RMAN, then user must activate the resource after the algorithm is activated and deactivate the resource before algorithm deactivation.

4.5 Algorithm Instance Deletion

Once filtering is complete, the test application must release the resources granted by the IRES resource Manager interface and delete the current algorithm instance. The following APIs are called in sequence:

1) `Free all resources granted by RMAN`
2) `algNumAlloc()` - To query the algorithm about the number of memory records it used.
3) `algFree()` - To query the algorithm to get the memory record information.

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

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

1) `RMAN_freeResources ()` - To free the resources that were allocated to the algorithm before process call.
2) `RMAN_exit()` - To delete the generic IRES RMAN and release memory.

4.6 Frame Buffer Management

4.6.1 Input and Output Frame Buffer

The algorithm has input buffers that stores frames until they are processed. These buffers at the input level are associated with a buffer inBufIDs. The output buffers are similarly associated with outBufIDs. The IDs are required to track the buffers that have been processed or locked. The algorithm uses this ID, at the end of the process call, to inform back to application whether it is a free buffer or not. Any buffer given to the algorithm should be considered locked by the algorithm, unless the buffer is returned to the application through `IVIDNF1_OutArgs->freeInBufID[] and IVIDNF1_OutArgs->freeOutBufID[]`.

For example,

Process Call #	1	2	3	4	5
inBufID	1	2	3	4	5
freeInBufID	1	2	3	4	5
outBufID	1	2	3	4	5
freeOutBufID	0	1	2	3	4

As shown in the table, one output buffer is locked by the algorithm when numPastRef 1. The input buffer gets freed immediately when the numFutureRef is zero

4.6.2 Frame Buffer Format

Algorithm expects un-compressed input in YUV semi planar format. MCTNF algorithm supports YUV 420 input format. Bit depth of each pixel can only be 8 bit.

5 API Reference

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

5.1 *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 provided.

Table 4 List of Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
`IVIDEO_ContentType`	`IVIDEO_CONTENTTYPE_NA`	Frame type is not available.
	`IVIDEO_PROGRESSIVE` `IVIDEO_PROGRESSIVE_FRAME` `IVIDEO_CONTENTTYPE_DEFAULT`	Progressive video content. Default value is `IVIDEO_PROGRESSIVE`
	`IVIDEO_INTERLACED` `IVIDEO_INTERLACED_FRAME`	Interlaced video content.
	`IVIDEO_INTERLACED_TOPFIELD`	Interlaced video content, top field.
	`IVIDEO_INTERLACED_BOTTOMFIELD`	Interlaced video content, bottom field.
`IVIDEO_OutputFrameStatus`	`IVIDEO_FRAME_NOERROR` `IVIDEO_OUTPUTFRAMESTATUS_DEFAULT`	Output buffer is available (default value). Default status of the output frame.
	`IVIDEO_FRAME_NOTAVAILABLE`	Algorithm does not have any output buffers.
	`IVIDEO_FRAME_ERROR`	Output buffer is available and corrupted.
`IVIDEO_VideoLayout`	`IVIDEO_FIELD_INTERLEAVED`	Buffer layout is interleaved.
	`IVIDEO_FIELD_SEPARATED`	Buffer layout is field separated.
	`IVIDEO_TOP_ONLY`	Buffer contains only top field.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`IVIDEO_BOTTOM_ONLY`	Buffer contains only bottom field.
`IVIDEO_BitRange`	`IVIDEO_YUVRANGE_FULL`	Pixel range for YUV is 0-255.
	`IVIDEO_YUVRANGE_ITU`	Pixel range for YUV is as per ITU-T .
`IVIDEO_DataMode`	`IVIDEO_FIXEDLENGTH`	Data is exchanged at interval of fixed size. This feature is not supported in this version of the algorithm
	`IVIDEO_SLICEMODE`	Slice mode.
	`IVIDEO_NUMROWS`	Number of rows, each row is 16 lines of video.
	`IVIDEO_ENTIREFRAME`	Processing of entire frame data.
`XDM_AccessMode`	`XDM_ACCESSMODE_READ`	Algorithm read from the buffer using the CPU.
	`XDM_ACCESSMODE_WRITE`	Algorithm writes to the buffer using the CPU.
`XDM_CmdId`	`XDM_GETSTATUS`	Query algorithm instance to fill `Status` structure.
	`XDM_SETPARAMS`	Set run-time dynamic parameters through the `DynamicParams` structure.
	`XDM_RESET`	Reset the algorithm. All fields in the internal data structures are reset and all internal buffers are flushed.
	`XDM_SETDEFAULT`	Restore the algorithm's internal state to its original, default values. The application needs to initialize the `dynamicParams.size` and `status.size` fields prior to calling `control()` with `XDM_SETDEFAULT`. The algorithm must write to the `status.extendedError` field, and potentially algorithm specific, extended fields. `XDM_SETDEFAULT` differs from `XDM_RESET`. In addition to restoring the algorithm's internal state, `XDM_RESET` also resets any channel related state.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_FLUSH	Handle end of stream conditions. This command forces the algorithm to output data without additional input. The recommended sequence is to call the control() function (with XDM_FLUSH) followed by repeated calls to the process() function until it returns an error. The algorithm should return the appropriate, class-specific EFAIL error when flushing is complete.
	XDM_GETBUFINFO	Query algorithm instance regarding its properties of input and output buffers. The application only needs to initialize the dynamicParams.size, the status.size, and set any buffer descriptor fields (example, status.data) to NULL prior to calling control() with XDM GETBUFINFO.
	XDM_GETVERSION	Query the algorithm's version. The result is returned in the data field of the respective _Status structure. There is no specific format defined for version returned by the algorithm. The memory is not allocated by algorithm and needs to be allocated by user. The buffer requirement for holding version number is of length IMCTVNF_VERSION_LENGTH Not supported in this version.
	XDM_GETCONTEXTINFO	Query a split MCTNF part for its context needs. Only split MCTNFs are required to implement this command. Not supported in this version.
	XDM_GETDYNPARAMSDEFAULT	Query the algorithm to fill the default values for the parameters, which can be configured dynamically. To get the current value of an algorithm instance's dynamic parameters, it is recommended that the algorithm provides them through the XDM GETSTATUS call.
	XDM_SETLATEACQUIREARG	Set an algorithm's 'late acquire' argument. Only algorithms that utilize the late acquire IRES feature may implement this command.
XDM_DataFormat	XDM_BYTE	Big endian stream (default value)

Group or Enumeration Class	**Symbolic Constant Name**	**Description or Evaluation**
	`XDM_LE_16`	16-bit little endian stream. Not supported in this version.
	`XDM_LE_32`	32-bit little endian stream. Not supported in this version.
	`XDM_LE_64`	64-bit little endian stream. Not supported in this version.
	`XDM_BE_16`	16-bit big endian stream. Not supported in this version.
	`XDM_BE_32`	32-bit big endian stream. Not supported in this version.
	`XDM_BE_64`	64-bit big endian stream. Not supported in this version.
`XDM_ChromaFormat`	`XDM_CHROMA_NA`	Chroma format not applicable.
	`XDM_YUV_420P`	YUV 4:2:0 planar. Not supported in this verison.
	`XDM_YUV_422P`	YUV 4:2:2 planar. Not supported in this version.
	`XDM_YUV_422IBE`	YUV 4:2:2 interleaved (big endian). Not supported in this version.
	`XDM_YUV_422ILE`	YUV 4:2:2 interleaved (little endian) Not supported in this version.
	`XDM_YUV_444P`	YUV 4:4:4 planar. Not supported in this verison.
	`XDM_YUV_411P`	YUV 4:1:1 planar. Not supported in this version.
	`XDM_GRAY`	Gray format. Not supported in this version.
	`XDM_RGB`	RGB color format. Not supported in this version.
	`XDM_YUV_420SP`	YUV 4:2:0 chroma semi-planar format (first plane is luma and second plane is CbCr interleaved) Default value.
	`XDM_ARGB8888`	Alpha plane color format. Not supported in this version.
	`XDM_RGB555`	RGB555 color format. Not supported in this version.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_RGB565	RGB565 color format. Not supported in this version.
	XDM_YUV_444ILE	YUV 4:4:4 interleaved (little endian) color format. Not supported in this version.
XDM_ErrorBit	XDM_PARAMSCHANGE	Bit 8 1 - Sequence Parameters Change 0 - Ignore This error is applicable for transcoders. It is set when some key parameter of the input sequence changes. The transcoder returns after setting this error field and the correct input sequence parameters are updated in outArgs.
	XDM_APPLIEDCONCEALMENT	Bit 9 1 - Applied concealment 0 - Ignore This error is applicable for decoders. It is set when the decoder was not able to decode the bit-stream, and the decoder has concealed the bit-stream error and produced the concealed output.
	XDM_INSUFFICIENTDATA	Bit 10 1 - Insufficient input data 0 - Ignore This error is applicable for decoders. This is set when the input data provided is not sufficient to produce one frame of data. This can be also be set for encoders when the number of valid samples in the input frame is not sufficient to process a frame.
	XDM_CORRUPTEDDATA	Bit 11 1 - Data problem/corruption 0 - Ignore This error is applicable for decoders. This is set when the bit-stream has an error and not compliant to the standard syntax.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`XDM_CORRUPTEDHEADER`	Bit 12 1 - Header problem/corruption 0 - Ignore This error is applicable for decoders. This is set when the header information in the bit-stream is incorrect. For example, it is set when Sequence, Picture, Slice, and so on are incorrect in video decoders.
	`XDM_UNSUPPORTEDINPUT`	Bit 13 1 – Un-supported feature/parameter in input 0 - Ignore This error is set when the algorithm is not able process a certain input data/bit-stream format. It can also be set when a subset of features in a standard are not supported by the algorithm. For example, if a video encoder only supports 4:2:2 formats, it can set this error for any other type of input video format.
	`XDM_UNSUPPORTEDPARAM`	Bit 14 1 - Unsupported input parameter or configuration 0 - Ignore This error is set when the algorithm does not support certain configurable parameters. For example, if the video decoder does not support the display width feature, it will return `XDM_UNSUPPORTEDPARAM` when the control function is called for setting the display width attribute.
	`XDM_FATALERROR`	Bit 15 1 - Fatal error (stop encoding) 0 - Recoverable error If there is an error, and this bit is not set, the error is recoverable. This error is set when the algorithm cannot recover from the current state. It informs the system not to try the next frame and possibly delete the multimedia algorithm instance. It implies the MCTNF will not work when reset. You should delete the current instance of the MCTNF.

Group or Enumeration Class	**Symbolic Constant Name**	**Description or Evaluation**
`XDM_MemoryType`	`XDM_MEMTYPE_ROW`	Linear (standard) memory. Deprecated.
	`XDM_MEMTYPE_RAW`	Linear (standard) memory.
	`XDM_MEMTYPE_TILED8`	
	`XDM_MEMTYPE_TILED16`	
	`XDM_MEMTYPE_TILED32`	
	`XDM_MEMTYPE_TILEDPAGE`	

Note:

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

Bit 16-32: Used for algorithm specific error codes.

Bit 0-7: Algorithm and implementation specific

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

Table 5 MCTNF Algorithm Error Status

Group or Enumeration Class	**Symbolic Constant Name**	**Description or Evaluation**
`IMCTNF_ErrorBit`	`IMCTNF_IMPROPER_HW_STATE`	Bit 16 - Device is not proper state to use.
	`IMCTNF_UNSUPPORTED_DEVICE`	Bit 18 – Feature non supported on this device.
	`IMCTNF_IMPROPER_DATASYNC_SETTING`	Bit 19 - data synch settings are not proper This error is set when algorithm is asked to operate at sub frame level but the call back function pointer is NULL

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	`IMCTNF_UNSUPPORTED_VIDNF1PARAMS`	Bit 20 - Invalid vidnf1 parameters This error is set when any parameter of struct IVIDNF1_Params is not in allowed range
	`IMCTNF_UNSUPPORTED_MOTIONSEARCHPARAMS`	Bit 21 - Invalid motion search parameters This error is set when any parameter of struct IMCTNF_MotionSearchParams is not in allowed range
	`IMCTNF_UNSUPPORTED_FIELDSELECTIONTYPE`	Bit 23 - Invalid field selection parameters This error is set when any parameter of struct IMCTNF_IntraCodingParams is not in allowed range
	`IMCTNF_UNSUPPORTED_NOISEFILTERPARAMS`	Bit 26 - Invalid noise filter related parameters This error is set when any parameter of struct IMCTNF_NoiseFilterParams is not in allowed range
	`IMCTNF_UNSUPPORTED_MCTNFPARAMS`	Bit 29 - Invalid Create time extended parameters This error is set when any parameter of struct IMCTNF_Params is not in allowed range
	`IMCTNF_UNSUPPORTED_VIDNF1DYNAMICPARAMS`	Bit 30 - Invalid base class dyanmic paaremeters during control This error is set when any parameter of struct IVIDNF1_DynamicParams is not in allowed range
	`IMCTNF_UNSUPPORTED_MCTNFDYNAMICPARAMS`	Bit 31 - Invalid exteded class dyanmic paaremeters during control This error is set when any parameter of struct IMCTNF_DynamicParams (excluding embedded structures) is not in allowed range

5.2 Data Structures

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

5.2.1 Common XDM Data Structures

This section includes the following common XDM data structures:

- `XDM1_SingleBufDesc`
- `XDM1_AlgBufInfo`
- `IVIDEO2_BufDesc`

- IVIDNF1_Fxns
- IVIDNF1_Params
- IVIDNF1_DynamicParams
- IVIDNF1_InArgs
- IVIDNF1_Status
- IVIDNF1_OutArgs

5.2.1.1 XDM1_SingleBufDesc

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
*buf	XDAS_Int8	Input	Pointer to the buffer address
bufSize	XDM2_BufSize	Input	Buffer size for tile memory/row memory
accessMask	XDAS_Int32	Input	Mask filled by the algorithm, declaring how the buffer was accessed by the algorithm processor. If the buffer was not accessed by the algorithm processor (for example, it was filled through DMA or other hardware accelerator that does not write through the algorithm's CPU), then bits in this mask should not be set. It is acceptable to set several bits in this mask, if the algorithm accessed the buffer in several ways. This mask is often used by the application and/or framework to manage cache on cache-based systems. See XDM AccessMode enumeration .

5.2.1.2 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	Output	Minimum number of input buffers
minNumOutBufs	XDAS_Int32	Output	Minimum number of output buffers
minInBufSize[XDM_MAX_IO_BUFFERS]	XDM2_BufSize	Output	Minimum size required for each input buffer

Field	Data Type	Input/ Output	Description
`minOutBufSize[XDM_MAX_IO_BUFFERS]`	`XDM2_BufSize`	Output	Minimum size required for each output buffer
`inBufMemoryType[XDM_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	Required memory type for each input buffer. See `XDM_MemoryType` enumeration .
`outBufMemoryType[XDM_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	Required memory type for each output buffer. See `XDM_MemoryType` enumeration details
`minNumBufSets`	`XDAS_Int32`	Output	Minimum number of buffer sets for buffer management

Note:

For MCTNF algorithm, the buffer details are:

- Number of input buffer required is 2 for YUV 420SP (`memType` is `XDM_MEMTYPE_RAW`)
- In video noise filter as the input buffer gets affected, the address of the output buffer must be given the same as input buffer.

The input buffer sizes (in bytes) for CIF format for 420SP chroma format is:

- Y buffer = 352 * 288
- CbCr buffer = 352 * 144

With horizontal and vertical search range of 64 pixels the sizes are,

- Y buffer = (64 + 352 + 64) * (64 + 288 + 64)
- CbCr buffer = (64 + 352 + 64) * (32 + 144 + 32)

When the input frame buffer that getting filtered by algorithm is not same as capture buffer then algorithm still returns the size of the buffer accessed by him. In these situations application should take care of proper buffer allocation for input frame buffer.

5.2.1.3 IVIDEO2_BufDesc

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
`numPlanes`	`XDAS_Int32`	Input/Output	Number of buffers for video planes

Field	Data Type	Input/ Output	Description
numMetaPlanes	XDAS_Int32	Input/Output	Number of buffers for metadata
dataLayout	XDAS_Int32	Input/Output	Video buffer layout, Only field interleaved is supported in the present version. See IVIDEO_VideoLayout enumeration for more details
planeDesc [IVIDEO_MAX_NUM_PLANES]	XDM2_SingleBufDesc	Input/Output	Description for video planes
metadataPlaneDesc [IVIDEO_MAX_NUM_METADATA_PLANES]	XDM2_SingleBufDesc	Input/Output	Description for metadata planes
secondFieldOffsetWidth[IVIDEO_MAX_NUM_PLANES]	XDAS_Int32	Input/Output	Offset value for second field in planeDesc buffer (width in pixels) Valid only if pointer is not NULL.
secondFieldOffsetHeight[IVIDEO_MAX_NUM_PLANES]	XDAS_Int32	Input/Output	Offset value for second field in planeDesc buffer (height in lines) Valid only if pointer is not NULL.
imagePitch[IVIDEO_MAX_NUM_PLANES]	XDAS_Int32	Input/Output	Image pitch for each plane
imageRegion	XDM_Rect	Input/Output	Decoded image region including padding/encoder input image (top left and bottom right).
activeFrameRegion	XDM_Rect	Input/Output	Actual display region/capture region (top left and bottom right).
extendedError	XDAS_Int32	Input/Output	Indicates the error type, if any. Not applicable for encoders.
frameType	XDAS_Int32	Input/Output	Video frame types. See enumeration IVIDEO_FrameType enumeration for more details. Not applicable for encoder input buffer.
topFieldFirstFlag	XDAS_Int32	Input/Output	Indicates when the application (should display)/(had captured) the top field first. Not applicable for progressive content. Not applicable for encoder reconstructed buffers. Valid values are XDAS_TRUE and XDAS_FALSE.
repeatFirstFieldFlag	XDAS_Int32	Input/Output	Indicates when the first field should be repeated. Valid values are XDAS_TRUE and XDAS_FALSE.

Field	Data Type	Input/ Output	Description
			Only applicable for interlaced content, not progressive. Not applicable for encoders.
frameStatus	XDAS_Int32	Input/Output	Video in/out buffer status. Not applicable for encoder reconstructed buffers. Not applicable for encoder input buffers.
repeatFrame	XDAS_Int32	Input/Output	Number of times the display process needs to repeat the displayed progressive frame. This information is useful for progressive content when the decoder expects the display process to repeat the displayed frame for a certain number of times. This is useful for pull-down (frame/field repetition by display system) support where the display frame rate is increased without increasing the decode frame rate. Default value is 0. Not applicable for encoder reconstructed buffers. Not required for encoder input buffer
contentType	XDAS_Int32	Input/Output	Video content type. See IVIDEO_ContentType enumeration for more details. This is useful when the content is both interlaced and progressive. The display process can use this field to determine how to render the display buffer.
chromaFormat	XDAS_Int32	Input/Output	Chroma format for encoder input data/decoded output buffer. See XDM_ChromaFormat enumeration for more details..
scalingWidth	XDAS_Int32	Input/Output	Scaled image width for post processing for decoder. Not applicable for encoders.
scalingHeight	XDAS_Int32	Input/Output	Scaled image height for post processing for decoder. Not applicable for encoders.
rangeMappingLuma	XDAS_Int32	Input/Output	Applicable for VC1, set to -1 as default for other MCTNFs
rangeMappingChroma	XDAS_Int32	Input/Ou	Applicable for VC1, set to -1 as

Field	Data Type	Input/ Output	Description
		tput	default for other MCTNFs
`enableRangeReductionFlag`	`XDAS_Int32`	Input/Output	Flag indicating whether to enable range reduction or not. Valid values are `XDAS_TRUE` and `XDAS_FALSE`. Applicable only for VC-1

5.2.1.4 IVIDNF1_Fxns

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
`ialg`	`IALG_Fxns`	Input	Structure containing pointers to all the XDAIS interface functions. For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).
`*process`	`XDAS_Int32`	Input	Pointer to the `process()` function. See section 0 for more information
`*control`	`XDAS_Int32`	Input	Pointer to the `control()` function. See section 0 for more information

5.2.1.5 IVIDNF1_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. For the default and supported values, see ***Table 6 Default and Supported Values for IVIDNF1_Params*****Error! Reference source not found.**.

‖ Fields

Field	Data Type	Input/ Output	Description
`size`	`XDAS_Int32`	Input	Size of the base or extended (if being used) data structure in bytes. Currently base class is not supported. Supported Values: `sizeof(IMCTVNF Params)`
`filterPreset`	`XDAS_Int32`	Input	Noise filter preset value. Only a default value of 0 is supported.
`maxHeight`	`XDAS_Int32`	Input	Maximum video height to be supported in pixels.
`maxWidth`	`XDAS_Int32`	Input	Maximum video width to be supported in pixels.
`dataEndianness`	`XDAS_Int32`	Input	Endianness of output data. See `XDM_DataFormat` enumeration in for more details.
`inputChromaFormat`	`XDAS_Int32`	Input	Chroma format for the input buffer. See `XDM_ChromaFormat` enumeration for more details.
`inputContentType`	`XDAS_Int32`	Input	Video content type of the buffer being encoded. See `IVIDEO_ContentType` enumeration for more details.
`inputDataMode`	`XDAS_Int32`	Input	If a sub-frame mode is provided the application must call VIDNF1_Fxns::control() with #XDM_SETPARAMS id prior to #IVIDNF1_Fxns::process() to set the IVIDNF1_DynamicParams::getDataFxn And IVIDNF1_DynamicParams::getDataHandle. Else, the alg can return error.

Field	Data Type	Input/ Output	Description
`outputDataMode`	`XDAS_Int32`	Input	If a sub-frame mode is provided the application must call IVIDNF1_Fxns::control() with #XDM_SETPARAMS id prior to #IVIDNF1_Fxns::process() to set the IVIDNF1_DynamicParams::putDataFxn, IVIDNF1_DynamicParams::putDataHandle (and optionally IVIDNF1_DynamicParams::getBufferFxn, and IVIDNF1_DynamicParams::getBufferHandle) .Else, the alg can return error.
`numInputDataUnits`	`XDAS_Int32`	Input	Number of input slices/rows.
`numOutputDataUnits`	`XDAS_Int32`	Input	Number of output slices/rows

5.2.1.6 IVIDNF1_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. For the default and supported values, see ***Table 7 Default and Supported Values for IVIDNF1_DynamicParams***

‖ Fields

Field	Data Type	Input/ Output	Description
`size`	`XDAS_Int32`	Input	Size of the basic or extended (if being used) data structure in bytes
`inputHeight`	`XDAS_Int32`	Input	Height of the input frame
`inputWidth`	`XDAS_Int32`	Input	Width of the input frame
`captureWidth`	`XDAS_Int32`	Input	DEFAULT(0): use imagewidth as pitch else use given capture width for pitch provided it is greater than image width.
`numPastRef`	`XDAS_Int32`	Input	Number of past reference frames used for motion search
`numFutureRef`	`XDAS_Int32`	Input	Number of future reference frames used for motion search

Field	Data Type	Input/ Output	Description
`putDataFxn`	`XDM_DataSyncPutFxn`	Input	Refer to XDM_DataSyncPutFxn
`putDataHandle`	`XDM_DataSyncHandle`	Input	Refer to XDM_DataSyncHandle
`getDataFxn`	`XDM_DataSyncGetFxn`	Input	Refer to XDM_DataSyncGetFxn
`getDataHandle`	`XDM_DataSyncHandle`	Input	Refer to XDM_DataSyncHandle
`getBufferFxn`	`XDM_DataSyncGetBufferFxn`	Input	Refer to XDM_DataSyncGetBufferFxn
`getBufferHandle`	`XDM_DataSyncHandle`	Input	Refer to XDM_DataSyncHandle
`lateAcquireArg`	`XDAS_Int32`	Input	Argument used during late acquire.

Note:

The following are the limitations on the parameters of `IVIDNF1_DynamicParams` data structure:

`inputHeight` <= `maxHeight`

`inputWidth` <= `maxWidth`

5.2.1.7 IVIDNF1_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 basic or extended (if being used) data structure in bytes.
`inBufID`	`XDAS_Int32`	Input	Identifier to attach with the corresponding input frames to be filtered. Zero (0) is not a supported `inBufID`. This value is reserved for cases when there no input buffer is provided. This is useful when frames require buffering and to support buffer management.
`outBufID`	`XDAS_Int32`	Input	Identifier to attach with the corresponding output frames to be filtered. Zero (0) is not a supported `outBufID`. This value is

Field	Data Type	Input/ Output	Description
			reserved for cases when there no output buffer is provided. This is useful when frames require buffering and to support buffer management.

5.2.1.8 IVIDNF1_Status

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
`size`	`XDAS_Int32`	Input	Size of the basic or extended (if being used) data structure in bytes.
`extendedError`	`XDAS_Int32`	Output	Extended error code. See `XDM_ErrorBit` enumeration for details.
`data`	`XDM1_SingleBufDesc`	Output	Buffer descriptor for data passing If this field is not used, the application must set `data.buf` to NULL. This buffer can be used as either input or output, depending on the command. The buffer will be provided by the application, and returned to the application on return of the `IVIDENC1_Fxns.control()` call. The algorithm must not retain a pointer to this data.
`filterPreset`	`XDAS_Int32`	Output	Filtering preset.
`inputChromaFormat`	`XDAS_Int32`	Output	Chroma format for the input buffer. See `XDM_ChromaFormat` enumeration for details.
`inputContentType`	`XDAS_Int32`	Output	Video content type of the buffer being encoded. See `IVIDEO_ContentType` enumeration for details.
`inputDataMode`	`XDAS_Int32`	Output	Input data mode

Field	**Data Type**	**Input/ Output**	**Description**
`outputDataMode`	`XDAS_Int32`	Output	Output data mode
`numInputDataUnits`	`XDAS_Int32`	Output	Number of input slices/rows.
`numOutputDataUnits`	`XDAS_Int32`	Output	Number of output slices/rows.
`configurationID`	`XDAS_Int32`	Output	Configuration ID of given MCTNF implementation.
`bufInfo`	`XDM1_AlgBufInf`	Output	This field provides the application with the algorithm's buffer requirements. The requirements may vary depending on the current configuration of the algorithm instance.
`nfDynamicParams`	`IVIDNF1_DynamicParams`	Input	Algorithm dynamic params.
`startX`	`XDAS_Int32`	Output	startX points to the filtered data in the output buffer provided by the application, along the X direction. The application would be providing a buffer bigger in size to the input resolution of the frame. This is to accomodate padding pixels. This will be populated after GETBUF_INFO control call.
`startY`	`XDAS_Int32`	Output	startY points to the filtered data in the output buffer provided by the application, along the Y direction. The application would be providing a buffer bigger in size to the input resolution of the frame. This is to accomodate padding pixels. This will be populated after GETBUF_INFO control call.

5.2.1.9 IVIDNF1_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`	Input	Size of the basic or extended (if being used) data structure in bytes.
`extendedError`	`XDAS_Int32`	Output	Extended error code. See `XDM ErrorBit` enumeration for details.
`displayID [IVIDEO2_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	Output ID corresponding to displayBufs[] A value of (0) indicates an invalid ID. The first zero entry in array will indicate end of valid outputIDs witihin the array. Hence the application can stop reading the array when it encounters the first zero entry.
`freeInBufID [IVIDEO2_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	ID corresponding to inBufID[] A value of (0) indicates an invalid ID. The first zero entry in array will indicate end of valid freeInBufIDs witihin the array. Hence the application can stop reading the array when it encounters the first zero entry.
`freeOutBufID [IVIDEO2_MAX_IO_BUFFERS]`	`XDAS_Int32`	Output	ID corresponding to outBufID[] A value of (0) indicates an invalid ID. The first zero entry in array will indicate end of valid freeOutBufIDs witihin the array. Hence the application can stop reading the array when it encounters the first zero entry.

5.2.2 MCTNF Algorithm Data Structures

This section includes the following MCTNF Algorithm specific extended data structures:

- ❑ `IMCTNF_Params`
- ❑ `IMCTNF_MotionSearchParams`
- ❑ `IMCTNF_NoiseFilterParams`
- ❑ `IMCTNF_DynamicParams`
- ❑ `IMCTNF_Inargs`
- ❑ `IMCTNF_Status`
- ❑ `IMCTNF_OutArgs`
- ❑ `IMCTNF_Fxns`

5.2.2.1 IMCTNF_Params

‖ Description

This structure defines the creation parameters and any other implementation specific parameters for a MCTNF Algorithm instance object. The creation parameters are defined in the XDM data structure, `IVIDNF1_Params`. For the default and supported values

Table 10 Default and Supported Values for IMCTNF_Params**Error! Reference source not found.**.

‖ Fields

Field	Data Type	Input/ Output	Description
`vidnf1Params`	`IVIDNF1_Params`	Input	See `IVIDNF1_Params` data structure for details.
`motionSearchParams`	`IMCTNF_MotionSearchParams`	Input	See IMCTNF_MotionSearchParams data structure for details.
`pConstantMemory`	`XDAS_Int32`	Input	User provided address for relocating constant memory section of library.
`debugTraceLevel`	`XDAS_UInt32`	Input	Debug Trace Level
`lastNFramesToLog`	`XDAS_UInt32`	Input	Last N frames log data to be maintained.
`enableAnalyticinfo`	`XDAS_Int8`	Input	Flag to enable Analytic Info. Currently not supported

Field	Data Type	Input/ Output	Description
`reservedParams`	`XDAS_Int32`	Input	Reserved parameters for MCTNF algorithm

5.2.2.2 IMCTNF_MotionSearchParams

‖ Description

This structure controls prameters for motion estimation. For the default and supported values, see ***Table 8 Default and Supported Values for IMCTNF_MotionSearchParams***.

‖ Fields

Field	Data Type	Input/ Output	Description
`motionSearchPreset`	`XDAS_Int8`	Input	Motion search preset. Currently not supported.
`searchRangeHor`	`XDAS_Int16`	Input	Horizontal search range for motion search
`searchRangeVer`	`XDAS_Int16`	Input	Vertical search range for motion search

5.2.2.3 IMCTNF_NoiseFilterParams

‖ Description

This structure controls parameters for main noise filtering process. For the default and supported values, see ***Table 9 Default and Supported Values for IMCTNF_NoiseFilterParams***.

‖ Fields

Field	Data Type	Input/ Output	Description
`meEffectivenessTh`	`XDAS_Int8`	Input	If ME distortion is higher than this threshold, then it indicates ME is finding it difficult to get match and it takes corrective action internally.
`blendingFactorQ4`	`XDAS_Int16`	Input	Blending factor in Q4. Higher the value more aggressive is filtering.

Field	Data Type	Input/ Output	Description
			0: Dynamic: Indicates MCTNF will internally choose value of minBlend or maxBlend based on content statistics
minBlendQ4	XDAS_Int32	Input	Minimum blending factor in Q4: 0: Dynamic: Indicates MCTNF will internally choose value of minBlendQ4 based on content statistics. minBlendQ4 should always be lower than maxBlendQ4.
maxBlendQ4	XDAS_Int32	Input	Maximum blending factor in Q4: 0: Dynamic: Indicates MCTNF will internally choose value of maxBlendQ4 based on content statistics. maxBlendQ4 should always be higher than minBlendQ4.
meLambdaType	XDAS_Int32	Input	Type of Lambda used for ME 0: fixed lambda as indicated by meLambdaFactorQ2, 1:dynamic Lambda, will be chosen dynamically between [meLambdaFactorQ2, maxLambdaQ2]
meLambdaFactorQ2	XDAS_Int32	Input	Lambda value used for ME in Q2 format
maxLambdaQ2	XDAS_Int32	Input	maximum value of lambda in case of dynamic lambda. This value is in Q2 format
sadForMinLambda	XDAS_Int32	Input	Upto this SAD minimum lambda is used, After that lambda is increased up to maxLambda.
fixWtForCurQ8	XDAS_Int32	Input	Fixed weighting factor in Q8 format, higher the value, higher the filtering, 0: dynamic. Not recommended for final product, useful for debugging
minWtForCurQ8	XDAS_Int32	Input	Min weight for current pixels for avoiding excessive filtering
biasZeroMotion	XDAS_Int32	Input	Bias for Zero MV over other MVs. 0: No Bias, 1: Positive Bias favoring Zero MV
staticMBThZeroMV	XDAS_Int32	Input	Static MB detection threshold for Zero MV MBs
staticMBThNonZeroMV	XDAS_Int32	Input	Static MB detection threshold for non Zero MV MBs
blockinessRemFactor	XDAS_Int32	Input	Removal factor for removing blocking artifacts which may be result of aggressive filtering. 0: disable, 1, 2 3, 4: Blockiness removal factor with different strengths, higher the number, more aggressive blockiness removal. Use of high strength may result in blurring. 5: indicates it will be chosen by MCTNF

Field	Data Type	Input/ Output	Description
			internally Currently 0,1,2,5 are only supported values.
sadForMaxStrength	XDAS_Int32	Input	Average Zero MV Luma SAD value above which max strength will be used. For lower values compared to sad4MaxStrength milder strength will be used.
mvXThForStaticLamda	XDAS_Int32	Input	Threshold in Q2 format for MVx to determine block level lambda. If local motion in x direction is above this threshold then smaller lambda will be used.
mvYThForStaticLamda	XDAS_Int32	Input	Threshold in Q2 format for MVy to determine block level lambda. . If local motion in y direction is above this threshold then smaller lambda will be used.

5.2.2.4 IMCTNF_NoiseFilterStats

‖ Description

This structure comprises of frame level statistics of various Noise Filter parameters

‖ Fields

Field	Data Type	Input/ Output	Description
avgZeroLumaSadSubBlk	XDAS_Int32	Output	Average Luma sad for the Mbs whose chosen motion vector was (0,0)
numMBWithZeroMV	XDAS_Int32	Output	Number of Mbs with zero motion vectors in current processed frame.
numStaticMBs	XDAS_Int32	Output	Number of Mb which were static in current processed frame. A Mb is decided to be static if motion around (3x3 Mb block centered at current Mb) that Mb is not significant.
avgYCbCrMECostSubBlock	XDAS_Int32	Output	Average Motion Estimation cost for luma chroma block combined for current frame
GMVx	XDAS_Int32	Output	Global motion vector (for x-direction) used for current frame
GMVy	XDAS_Int32	Output	Global motion vector (for y-direction) used for current frame

Field	Data Type	Input/ Output	Description
`maxBlendPic`	`XDAS_Int32`	Output	Maximum blending used in current picture.
`minBlendPic`	`XDAS_Int32`	Output	Minimum blending used in current picture.
`filtStrength`	`XDAS_Int32`	Output	Filter strength used for current frame.
`mePassed`	`XDAS_Int32`	Output	ME overall statistics. If this field is true then it indicates that ME was able to find reasonable good matches. If this field is zero then it indicates that ME was not able to find good match for Mbs due to high noise present.
`picLambdaQ2`	`XDAS_Int32`	Output	Lambda value for ME SAD penalization

5.2.2.5 IMCTNF_DynamicParams

‖ Description

This structure defines the run-time parameters and any other implementation specific parameters for a MCTNF Algorithm instance object. The run-time parameters are defined in the XDM data structure, `IVIDNF1_DynamicParams`. For the default and supported values, see ***Table 11 Default and Supported Values for IMCTNF_DynamicParams***

‖ Fields

Field	Data Type	Input/ Output	Description
`vidnf1DynamicParams`	`IVIDNF1_DynamicParams`	Input	See IVIDNF1_DynamicParams data structure for details.
`motionSearchParams`	`IMCTNF_MotionSearchParams`	Input	See IMCTNF_MotionSearchParams data structure for details.
`noiseFilterParams`	`IMCTNF_NoiseFilterParams`	Input	See IMCTNF_NoiseFilterParams data structure for details.
`searchCenter`	`XDM_Point`	input	Center point for motion estimation
`reservedDynParams`	`XDAS_Int32`	Input	Reserved parameters for future usages

5.2.2.6 IMCTNF_Inargs

‖ Description

This structure defines the run-time input arguments for MCTNF Algorithm instance object.

‖ Fields

Field	**Data Type**	**Input/ Output**	**Description**
`vidnf1InArgs`	`IVIDNF1_InArgs`	Input	See IVIDNF1_Inargs data structure for details

5.2.2.7 IMCTNF_Status

‖ Description

This structure defines parameters that describe the status of the MCTNF Algorithm and any other implementation specific parameters. The status parameters are defined in the XDM data structure, `IVIDNF1_Status`.

‖ Fields

Field	**Data Type**	**Input/ Output**	**Description**
`vidnf1Status`	`IVIDNF1_Status`	Output	See IVIDNF1_Status data structure for details.
`motionSearchParams`	`IMCTNF_MotionSearchParams`	Output	See IMCTNF_MotionSearchParams data structure for details.
`debugTraceLevel`	`XDAS_UInt32`	Output	debug trace level flag. From minimal trace to verbose.
`lastNFramesToLog`	`XDAS_UInt32`	Output	Last N Frames to Log
`enableAnalyticinfo`	`XDAS_Int8`	Output	Flag to enable analytic info
`searchCenter`	`XDM_Point`	Output	search Center for motion search
`extMemoryDebugTraceAddr`	`XDAS_UInt32`	Output	external memory address for dumping the debug trace
`extMemoryDebugTraceSize`	`XDAS_UInt32`	Output	external memory size of the debug trace buffer.

5.2.2.8 IMCTNF_OutArgs

‖ Description

This structure defines the run-time output parameters for the MCTNF Algorithm instance object.

‖ Fields

Field	Data Type	Input/ Output	Description
`vidnf1OutArgs`	`IVIDNF1_OutArgs`	Output	See IVIDNF1_OutArgs data structure for details.
`nfStats`	`IMCTNF_NoiseFilterStats`	Output	See IMCTNF_NoiseFilterStats data structure for details

5.2.2.9 IMCTNF_Fxns

‖ Description

This structure defines all the operations on MCTNF Algorithm instance objects.

‖ Fields

Field	Data Type	Input/ Output	Description
`ividnf1`	`IVIDNF1_Fxns`	Output	See IVIDNF1_Fxns data structure for details.

5.3 Default and Supported Values of Parameter

This section provides the default and supported values for the following data structures:

- `IVIDNF1_Params`
- `IVIDNF1_DynamicParams`
- `IMCTNF_MotionSearchParams`
- `IMCTNF_NoiseFilterParams`
- `IMCTNF_Params`
- `IMCTNF_DynamicParams`

Table 6 Default and Supported Values for IVIDNF1_Params

Field	Default Value	Supported Value
Size	sizeof(IMCTNF_Params)	❑ sizeof(IVIDNF1_Params) ❑ sizeof(IMCTNF_Params)
filterPreset	0	Only 0 is supported. High quality.
maxHeight	1920	[288, 2048] Note :: Height has to multiple of 2.
maxWidth	1088	[352, 2048] Note:: Width should be multiple of 16.
dataEndianness	XDM_BYTE	❑ XDM_BYTE
inputChromaFormat	XDM_YUV_420SP	XDM_YUV_420SP
inputContentType	IVIDEO_PROGRESSIVE	IVIDEO_PROGRESSIVE IVIDEO_INTERLACED is not supported at present.
inputDataMode	IVIDEO_ENTIREFRAME	IVIDEO_ENTIREFRAME. Note : Currently only full frame mode is supported. No sub frame level synchronization.
outputDataMode	IVIDEO_ENTIREFRAME	IVIDEO_ENTIREFRAME. Note : Currently only full frame mode is supported. No sub frame level synchronization.
numInputDataUnits	NA	NA Note :: dont care field in current scenario, as sub frame level synchronization is not supported currently.
numOutputDataUnits	NA	NA Note :: dont care field in current scenario, as sub frame level synchronization is not supported currently.

Table 7 Default and Supported Values for IVIDNF1_DynamicParams

Field	Default Value	Supported Value
`size`	`sizeof(IMCTNF_DynamicParams)`	❑ `sizeof(IVIDNF1_DynamicParams)` ❑ `sizeof(IMCTNF_DynamicParams)`
`inputHeight`	1088	[288, 1088] Height :: height should be multiple of 2.
`inputWidth`	1920	[352, 1920] Note :: Input width should be a multiple of 16
`captureWidth`	1920	Currently supported value is same as inputWidth.
`numPastRef`	1	Only supported Value is 1
`numFutureRef`	0	Only supported Value is 0
`putDataFxn`	NA	NA Note :: Not supported currently. Requires when sub frame level data communication is enabled.
`putDataHandle`	NA	NA Note :: Not supported currently. Requires when sub frame level data communication is enabled.
`getDataFxn`	NA	NA Note :: Not supported currently. Requires when sub frame level data communication is enabled.
`getDataHandle`	NA	NA Note :: Not supported currently. Requires when sub frame level data communication is enabled.
`getBufferFxn`	NA	NA Note :: Not supported currently. Requires when sub frame level data communication is enabled.
`getBufferHandle`	NA	NA Note :: Not supported currently. Requires when sub frame level data communication is enabled.
`lateAcquireArg`	Don’t Care	Any integer. Note :: Any integer value can be provided by this. MCTNF doesn’t use this value, it is pass through argument and used as one of the argument while doing call back function of “acquire” of IRES

Table 8 Default and Supported Values for IMCTNF_MotionSearchParams

Field	Default Value	Supported Value
`motionSearchPreset`	4	4 Not tested for other values.
`searchRangeHor`	144	144 Not tested for other values.
`searchRangeVer`	32	32 Not tested for other values.

Table 9 Default and Supported Values for IMCTNF_NoiseFilterParams

Field	Default Value	Supported Value
`meEffectivenessTh`	900	[400,3000]
`blendingFactorQ4`	0	{0, 8, 16, 24, 40}
`minBlendQ4`	0	{0, 8, 16, 24, 40}
`maxBlendQ4`	0	{0, 8, 16, 24, 40}
`meLambdaType`	1	[0,1]
`meLambdaFactorQ2`	96	[24, 255]
`maxLambdaQ2`	768	[24,255]
`sadForMinLambda`	200	[100,3000]
`fixWtForCurQ8`	0	[0,256]
`minWtForCurQ8`	41	[0,256]
`biasZeroMotion`	1	[0,1]
`staticMBThZeroMV`	8	[0,10]
`staticMBThNonZeroMV`	10	[0,10]
`blockinessRemFactor`	2	{0,1,2,5}
`sadForMaxStrength`	250	[100,3000]

Field	Default Value	Supported Value
`mvXThForStaticLamda`	0	[0, 256]
`mvYThForStaticLamda`	0	[0, 256]

Table 10 Default and Supported Values for IMCTNF_Params

Field	Default Value	Supported Value
`vidnfParams`	See Table 6 Default and Supported Values for IVIDNF1_Params	
`motionSearchParams`	See Table 8 Default and Supported Values for IMCTNF_MotionSearchParams	
`pConstantMemory`	NULL	Any valid address where Const can be located.
`debugTraceLevel`	0	[0,2]
`lastNFramesToLog`	0	
`enableAnalyticinfo`	0	0
`reservedParams`	NA	NA Note:: For future usages

Table 11 Default and Supported Values for IMCTNF_DynamicParams

Field	Default Value	Supported Value
`vidnfDynamicParams`	See Table 7 Default and Supported Values for IVIDNF1_DynamicParams	
`motionSearchParams`	See Table 8 Default and Supported Values for IMCTNF_MotionSearchParams	
`noiseFilterParams`	See Table 9 Default and Supported Values for IMCTNF_NoiseFilterParams	
`searchCenter`	0x7FFF, 0x7FFF	[-64,64]
`reservedDynParams`	0	NA

5.4 Interface Functions

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

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

You must call these APIs in the following sequence:

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

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

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

5.5 Creation APIs

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

‖ Name

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

‖ Synopsis

XDAS_Int32 algNumAlloc(Void);

‖ Arguments

Void

‖ Return Value

XDAS_Int32; /* number of buffers required */

‖ Description

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

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

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

|| See Also

algAlloc()

Name

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

|| Synopsis

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

|| Arguments

IALG_Params *params; /* algorithm specific attributes */

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

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

|| Return Value

XDAS_Int32 /* number of buffers required */

|| Description

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

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

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

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

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

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

|| See Also

algNumAlloc()
algFree()

5.6 Initialization API

Initialization API is used to initialize an instance of the algorithm. The initialization parameters are defined in the `IVIDNF1_Params` structure (see section 0 for details).

|| Name

`algInit()` – initialize an algorithm instance

|| Synopsis

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

|| Arguments

IALG_Handle handle; /* algorithm instance handle*/

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

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

IALG_Params *params; /*algorithm init parameters */

|| Return Value

IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */

|| Description

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

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

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

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

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

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

Since there is no mechanism to return extended error code for unsupported parameters, this version of algorithm returns `IALG_EOK` even if some parameter unsupported is set. But subsequence control/process call it returns the detailed error code

‖ See Also

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

5.7 Control API

Control API is used for controlling the functioning of the algorithm instance during run-time. This is done by changing the status of the controllable parameters of the algorithm during run-time. These controllable parameters are defined in the `Status` data structure (see section 0 for details).

‖ Name

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

‖ Synopsis

XDAS_Int32 (*control) (IVIDNF1_Handle handle, IVIDNF1_Cmd id, IVIDNF1_DynamicParams *params, IVIDNF1_Status *status);

‖ Arguments

IVIDNF1_Handle handle; /* algorithm instance handle */

IVIDNF1_Cmd id; /* algorithm specific control commands*/

IVIDNF1_DynamicParams *params /* algorithm run time parameters */

IVIDNF1_Status *status /* algorithm instance status parameters */

‖ Return Value

IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */

XDM_EUNSUPPORTED; /* status indicating parameters not supported*/

‖ Description

This function changes the run time parameters of an algorithm instance and queries the algorithm's `status.` `control()` must only be called after a successful `call to algInit()` and must never be called after a call `to algFree()`.

The first argument to control() is a handle to an algorithm instance.

The second argument is an algorithm specific control command. See XDM_CmdId enumeration for details.

The third and fourth arguments are pointers to the IVIDNF1_DynamicParams and IVIDNF1_Status data structures respectively.

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

|| Preconditions

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

- `control()` can only be called after a successful return from `algInit()` and `algActivate()`.
- If algorithm uses DMA resources, `control()` can only be called after a successful return from `DMAN3_init()`.
- `handle` must be a valid handle for the algorithm's instance object.
- params must not be NULL and must point to a valid `IVIDNF1_DynamicParams` structure.
- status must not be NULL and must point to a valid `IVIDNF1_Status` structure.
- If a buffer is provided in the `status->data` field, it must be physically contiguous and owned by the calling application.

|| Postconditions

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

- If the control operation is successful, the return value from this operation is equal to `IALG_EOK;` otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value. If status or handle is NULL then MCTNF returns `IALG_EFAIL`
- If the control command is not recognized or some parameters to act upon are not supported, the return value from this operation is not equal to `XDM_EUNSUPPORTED`.
- The algorithm should not modify the contents of params. That is, the data pointed to by this parameter must be treated as read-only.
- If a buffer was provided in the `status->data` field, it is owned by the calling application.

|| Example

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

|| See Also

algInit(), algActivate(), process()

5.8 Data Processing API

Data processing API is used for processing the input data.

|| Name

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

|| Synopsis

```
void algActivate(IALG_Handle handle);
```

|| Arguments

IALG_Handle handle; /* algorithm instance handle */

|| Return Value

Void

Description

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

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

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

|| See Also

`algDeactivate()`

‖ Name

`process()` – basic encoding/decoding call

‖ Synopsis

XDAS_Int32 (*process)(IVIDNF1_Handle handle, XDM1_BufDesc *inBufs, XDM1_BufDesc *outBufs, IVIDNF1_InArgs *inargs, IVIDNF1_OutArgs *outargs);

‖ Arguments

IVIDNF1_Handle handle; /* algorithm instance handle */

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

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

IVIDNF1_InArgs *inargs /* algorithm runtime input arguments */

IVIDNF1_OutArgs *outargs /* algorithm runtime output arguments */

‖ Return Value

IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */

‖ Description

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

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

The fourth argument is a pointer to the `IVIDNF1_InArgs` data structure that defines the run time input arguments for an algorithm instance object.

The last argument is a pointer to the `IVIDNF1_OutArgs` data structure that defines the run time output arguments for an algorithm instance object.

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

‖ Preconditions

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

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

- ❑ Buffer descriptor for input and output buffers must be valid.
- ❑ Input buffers must have valid input data.
- ❑ `inBufs->numBufs` indicates the total number of input
- ❑ Buffers supplied for input frame, and conditionally, the algorithms meta data buffer.
- ❑ `inArgs` must not be NULL and must point to a valid `IVIDNF1_InArgs` structure.
- ❑ `outArgs` must not be NULL and must point to a valid `IVIDNF1_OutArgs` structure.
- ❑ `inBufs` must not be NULL and must point to a valid `XDM1_BufDescIn` structure.
- ❑ `inBufs->bufDesc[0].bufs` must not be NULL, and must point to a valid buffer of data that is at least `inBufs->bufDesc[0].bufSize` bytes in length.
- ❑ `outBufs` must not be NULL and must point to a valid `XDM_BufDesc` structure.
- ❑ `outBufs->buf[0]` must not be NULL and must point to a valid buffer of data that is at least `outBufs->bufSizes[0]` bytes in length.
- ❑ The buffers in `inBuf` and `outBuf` are physically contiguous and owned by the calling application.

‖ Postconditions

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

- ❑ If the process operation is successful, the return value from this operation is equal to `IALG_EOK;` otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.
- ❑ After successful return from `process()` function, `algDeactivate()` can be called.
- ❑ The algorithm must not modify the contents of `inArgs`.
- ❑ The algorithm must not modify the contents of `inBufs`, with the exception of `inBufs.bufDesc[].accessMask`. That is, the data and buffers pointed to by these parameters must be treated as read-only.
- ❑ The algorithm must appropriately set/clear the `XDM1_BufDescIn::bufDesc[].accessMask` field in `inBufs` to indicate the mode in which each of the buffers in `inBufs` were read. For example, if the algorithm only read from `inBufs.bufDesc[0].buf` using the algorithm processor, it could utilize `#XDM_SETACCESSMODE_READ` to update the appropriate `accessMask` fields. The application may utilize these returned values to manage cache.
- ❑ The buffers in `inBufs` are owned by the calling application.

‖ Example

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

‖ See Also

algInit(), algDeactivate(), control()

Note:

The algorithm cannot be preempted by any other algorithm instance. That is, you cannot perform task switching while filtering of a particular frame is in progress. Pre-emption can happen only at frame boundaries and after `algDeactivate()` is called.

The input data is an uncompressed noisy video frame in one of the format defined by `inputChromaFormat` of `IVIDNF1_Params` structure. The output data is also uncompressed but a filtered version of the frame. The output is written in the same buffer supplied as input

`outBufs->bufs[0]` buffer may contain the luma filtered data.

`outBufs->bufs[1]` buffer may contain the CbCr inteleaved filtered data. See `IVIDNF1_OutArgs. displayBufs` for more details.

‖ Name

`algDeactivate()` – save all persistent data to non-scratch memory

‖ Synopsis

Void algDeactivate(IALG_Handle handle);

‖ Arguments

IALG_Handle handle; /* algorithm instance handle */

‖ Return Value

Void

‖ Description

`algDeactivate()` saves any persistent information to non-scratch buffers using the persistent memory that is part of the algorithm's instance object.

The first (and only) argument to `algDeactivate()` is an algorithm instance handle. This handle is used by the algorithm to identify various buffers that must be saved prior to next cycle of `algActivate()` and processing.

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

‖ See Also

`algActivate()`

5.9 Termination API

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

‖ Name

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

‖ Synopsis

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

‖ Arguments

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

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

‖ Return Value

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

‖ Description

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

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

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

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

‖ See Also

`algAlloc()`

6 Frequently Asked Questions

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

6.1 Code Build and Execution

Question	Answer

6.1.1 Algorithm Related

Question	Answer