Object Detection using TI's TMS320C66x DSP

User Guide

TEXAS INSTRUMENTS

June 2015

IMPORTANT NOTICE

1 Read This First

1.1 About This Manual

This document describes how to install and work with Texas Instruments' (TI) Object Detection Module implemented on TI's TMS320C66x DSP. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

TI's Object Detection Module implementations are based on IVISION interface. IVISION interface 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 vision and imaging algorithms with other software to build a high level vision system based on C66x DSP.

This document assumes that you are fluent in the C language, and aware of vision and image processing applications. Good knowledge of eXpressDSP Algorithm Interface Standard (XDAIS) 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 standards. It also provides an overview of Object Detection 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 Object Detection Module.

1.4 Related Documentation From Texas Instruments

This document frequently refers TI's DSP algorithm standards called XDAIS. To obtain a copy of document related to any of these standards, visit the Texas Instruments website at www.ti.com.

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
OBJDET	Object Detection Module
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

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 (Object Detection Module on TMS320C66x DSP) and version number. The version number of the Object Detection Module is included in the Title of the Release Notes that accompanies the product release.

1.8 Trademarks

Code Composer Studio, eXpressDSP, Object Detection Module are trademarks of Texas Instruments.

2 Introduction

This chapter provides a brief introduction to XDAIS. It also provides an overview of TI's implementation of Object Detection on the C66x DSP and its supported features.

2.1 *Overview of XDAIS*

TI's vision analytics applications are based on IVISION interface. IVISION 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).

2.2 Overview of Object Detection

The object detection module can be used to detect rigid and non rigid objects such as traffic signs and pedestrians. The module also performs traffic sign recognition and pedestrian tracking apart from just detection. The algorithm bundles a classifier, window grouping methods, object tracking methods and object recognition modules. It assumes that feature planes are provided as an input by another processor such as EVE or another DSP. The classifier used is Adaboost and it is trained with "HOG planes on Y with 6 bins + Y + U + V + gradient Magnitude" feature planes.

Information regarding feature planes generation is beyond the scope of this document.

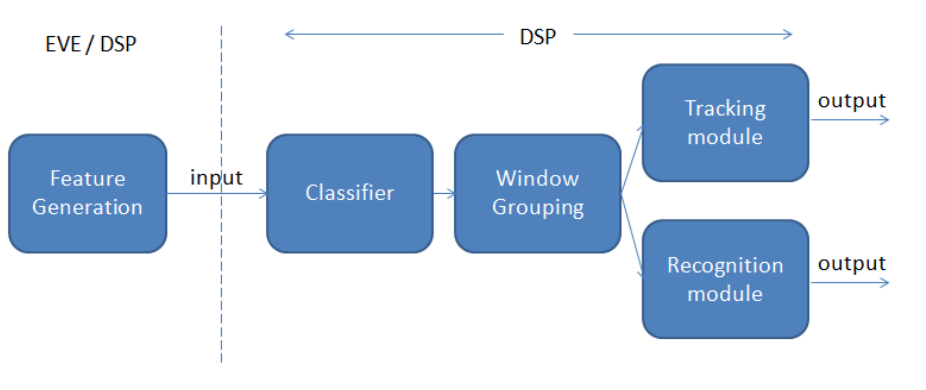

Figure 1 Fundamental blocks of Object Detection

2.3 Supported Services and Features

This user guide accompanies TI's implementation of Object Detection Algorithm on the TI's C66x DSP.

This version of the Object Detection has the following supported features of the standard:

- Supports 16 bit feature vectors.
- Supports Pedestrian Detection and tracking.
- Supports Traffic sign Detection and recognition.
- Supports upto 24 pyramid scales.
- Supports image resolution upto 1280x720
- Support for user control performance and quality knobs.
- Independent of any operating system.

This version of the Object Detection does not support following features:

- Traffic sign of any other country apart from Germany

3 Installation Overview

This chapter provides a brief description on the system requirements and instructions for installing Object Detection module. 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 TI's C66x DSP on TDA2x platform. The algorithm shall work on any future TDA platforms hosting C66x DSP.

3.1.2 Software

The following are the software requirements for the stand alone functioning of the Object Detection module:

- **Development Environment:** This project is developed using TI's Code Generation Tool 7.4.4. Other required tools used in development are mentioned in section 3.3
- The project are built using g-make (GNU Make version 3.81). GNU tools comes along with CCS installation.

3.2 Installing the Component

The algorithm component is released as install executable. 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 200.V.OD.C66x.00.03. Folder structure of this top level directory is shown in below figure.

After installing, set the environment variable “DSP_SW_ROOT” to the installed directory like <install directory>\200.V.OD.C66x.00.04\

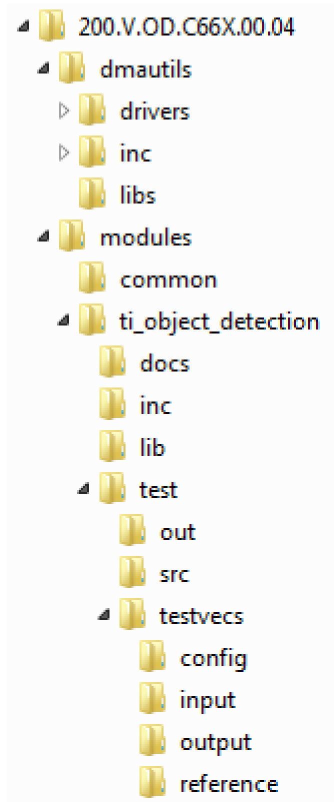

Figure 2 Component Directory Structure In case of Object Release

Table 2 Component Directories in case of Object release

Sub-Directory	Description
\dmautils	Top level folder for DSP/EDMA API’s
\dmautils\inc	Top level folder for DSP/EDMA API’s header files
\dmautils\libs	Contains dmautils library
\modules	Top level folder containing different DSP app modules
\modules\common	Common files for building different DSP modules
\modules \ti_object_detection	Object detection module for C66x DSP
\modules \ti_object_detection \docs	User guide and Datasheet for Object detection module

Sub-Directory	Description
\modules\ti_object_detection\inc	Contains iobjdet_ti.h interface file
\modules\ti_object_detection\lib	Contains Object detection algorithm library
\modules\ti_object_detection\test	Contains standalone test application source files
\modules\ti_object_detection\test\out	Contains test application .out executable
\modules\ti_object_detection\test\src	Contains test application source files
\modules\ti_object_detection\test\testvecs	Contains config, input, output, reference test vectors
\modules\ti_object_detection\test\testvecs\config	Contain config file to set various parameters exposed by Object detection module
\modules\ti_object_detection\test\testvecs\input	Contains sample input feature vector .bin file
\modules\ti_object_detection\test\testvecs\output	Contains output .txt file with a list of objects detected
\modules\ti_object_detection\test\testvecs\reference	Contains reference .txt file with a list of objects detected

3.3 Building Sample Test Application

This Object detection library has been accompanied by a sample test application. To run the sample test application XDAIS tools are required.

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

The version of the XDAIS required is 7.24.00.04

The version of the Code Generation tools required is 7.4.4

The version of EDMA3 Low Level Drivers required is 02.12.00.20

The version of C66x Mathlib required 3.1.0.0

3.3.1 Installing XDAIS tools (XDAIS)

XDAIS version 7.24 can be downloaded from the following website:

http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/xdais/

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_PATH`” pointing to <install directory>\<xdais_directory>

3.3.2 Installing Code Generation Tools

Install Code generation Tools version 7.4.4 from the link

https://www-a.ti.com/downloads/sds_support/TICodegenerationTools/download.htm

After installing the CG tools, set the environment variable to “`DSP_TOOLS`” to the installed directory like <install directory>\<cgtools_directory>

3.3.3 Installing EDMA3 Low level Drivers (LLD)

EDMA3 LLD 02.12.00.20 can be downloaded from the following website:

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_tii/psp/edma3_lld/index.html

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

C:\CCStudio5.0

Set a system environment variable named "`EDMA3_LLD_ROOT`" pointing to <install directory>\<edma3lld_directory>

3.3.4 Installing C66x Mathlib

Install C66x Mathlib version 3.1.0.0 from the link

http://software-dl.ti.com/sdoemb/sdoemb_public_sw/mathlib/latest/index_FDS.html

After installing Mathlib, set the environment variable to "`MATHLIB_INSTALL_DIR`" to the installed directory like <install directory>\<mathlib_directory>

3.3.5 Building the Test Application Executable through GMAKE

The sample test application that accompanies Object Detection module 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 as mentioned.
2) Verify that you have installed XDAIS as mentioned
3) Verify that appropriate environment variables have been set as discussed in this above sections.
4) Build the sample test application project by gmake
   a) modules\ti_object_detection\test> gmake clean
   b) modules\ti_object_detection\test> gmake all
5) The above step creates an executable file, test_object_detection_algo.out in the modules\ti_object_detection\test\out sub-directory.
6) Open CCS with TDA2x platform selected configuration file. Select Target > Load Program on C66x DSP, browse to the modules\ti_object_detection\test\out sub-directory, select the executable created in step 5, and load it into Code Composer Studio in preparation for execution.
7) Select Target > Run on C66x DSP window to execute the sample test application.
8) Sample test application takes the input files stored in the \test\testvecs\input sub-directory, runs the module.
9) The reference files stored in the \test\testvecs\reference sub-directory can be used to verify that the object detection is functioning as expected.

10) On successful completion, the test application displays the information for each feature frame and writes the information regarding the detected objects in the \test\testvecs\output sub-directory.

11) User should compare with the reference provided in \test\testvecs\reference directory. Both the content should be same to conclude successful execution.

3.4 Configuration File

This algorithm is shipped along with:

- Algorithm configuration file (object_detection.cfg) – specifies the configuration parameters used by the test application to configure the Algorithm.

3.4.1 Test Application Configuration File

The algorithm configuration file, object_detection.cfg contains the configuration parameters required for the algorithm. The object_detection.cfg file is available in the \test\testvecs\config sub-directory.

A sample object_detection.cfg file is as shown.

```
#------------------------------------------------------------------#
# Common Parameters                                                #
#------------------------------------------------------------------#
inFileName       = "../testvecs/input/00201_1280x720_420spl_10fr.bin"
outFileName      = "../testvecs/output/00201_1280x720_420spl_10fr_PD_TSR.txt"
maxImageWidth    = 1280  # Maximum width of the input image.
maxImageHeight   = 720  # Maximum height of the output image.
maxFrames        = 10   # Maximum number of input frames.
maxScales        = 28   # Maximum number of input scales to be checked. MAX_VALUE = 24

detectionMode    = 0    #(0-3), 0:(default) HIGH QUALITY check all points,
                        # 1: HIGH SPEED, skip every other point horizontally

roiPreset        = 0    # 0: Full frame processing
                        # 1: Dynamic ROI processing.

refreshInterval  = 0    # Valid only when roiPreset = 1,
                        # (default) A value of 0 will enable full frame processing for
all frames (F, F, F, F, ...)
                        # A value of 1 will enable full frame processing for every
other frame (F, R, F, R, ...)
                        # A value of 2 will enable full frame processing after two
frames (F, R, R, F, R, R, ...)
                        # And so on. Max value is 10.

#------------------------------------------------------------------#
```

```
# PD Parameters                                                    #
#------------------------------------------------------------------#
enablePD             =  1  # 0: Disable PD, 1: Enable PD
classifierTypePD     =  0  # 0: Adaboost
trackingMethodPD     =  1  # 0: Disabled, 1: Kalman Filter based
softCascadeThPD      = -1  # Soft cascade threshold for Adaboost
strongCascadeThPD    = -1  # Strong cascade threshold for Adaboost

#------------------------------------------------------------------#
# TSR Parameters                                                   #
#------------------------------------------------------------------#
enableTSR              =  1 # 0: Disable TSR, 1: Enable TSR
classifierTypeTSR      =  0 # 0: Adaboost
trackingMethodTSR      =  0 # 0: Disabled, 1: Kalman Filter based
recognitionMethodTSR   =  1 # 0: Disabled, 1: LDA
softCascadeThTSR       = -1 # Soft cascade threshold for Adaboost
strongCascadeThTSR     =  7 # Strong cascade threshold for Adaboost
```

If you specify additional fields in the object_detection.cfg file, ensure that you modify the test application appropriately to handle these fields.

3.5 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 Object Detection component.

4.1 Overview of the Test Application

The test application exercises the `IVISION` and extended class of the Object Detection library. The source files for this application are available in the \test\src sub-directories.

Test Application	XDAIS – IVISION interface	DSP Apps
Algorithm instance creation and initialization	--------- algNumAlloc() ---------> --------- algAlloc() ---------> --------- algInit() --------->	
Process Call	--------- control() ---------> --------- process() ---------> --------- control() --------->	
Algorithm instance deletion	--------- algNumAlloc() ---------> --------- algFree() --------->	

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, object detection requires parameters such as maximum image height, maximum image 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 object_detection.cfg and reads the various configuration parameters required for the algorithm.

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

2) Sets the `TI_OD_CreateParams` 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 feature planes into the application input buffer and makes a process call

5) For each frame dumps out the detected points along with meta data to specified output 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()`:

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

7) `algAlloc()` - To query the algorithm about the memory requirement to be filled in the memory records.

8) `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.

IMPORTANT! In this release, the algorithm assumes a fixed number of EDMA channels and does not rely on any IRES resource allocator to allocate the physical EDMA channels. This EDMA channel allocation method will be moved to IRES based mechanism in subsequent releases.

IMPORTANT! In this release, the algorithm requests two types of internal memory via IALG_DARAM0 and IALG_DARAM1 enums. The performance of the algorithm is validated by allocating DARAM0 to L1D SRAM and DARAM1 to L2 SRAM. Refer datasheet for more information regarding data and program memory sizes.

4.4 Process Call

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

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

10) 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 `IALG_GETBUFINFO` command.

11) Calls the `process()` function to detect objects in the provided feature plane. The inputs to the process function are input and output buffer descriptors, pointer to the `IVISION_InArgs` and `IVISION_OutArgs` structures.

12) When the `process()` function is called, 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:

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

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

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

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

17) `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 `process` is complete, the test application must release the resources granted by the IRES resource Manager interface if any and delete the current algorithm instance. The following APIs are called in sequence:

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

19) `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.

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 bufferId mentioned in input buffer descriptor. The output buffers are similarly associated with bufferId mentioned in the output buffer descriptor. 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 `IVISION_OutArgs->inFreeBufID[] and IVISION_OutArgs->outFreeBufID[]`.

For example,

Process Call #	1	2	3	4	5
bufferID (input)	1	2	3	4	5
bufferID (output)	1	2	3	4	5
inFreeBufID	1	2	3	4	5
outFreeBufID	1	2	3	4	5

The input buffer and output buffer is freed immediately once process call returns.

4.6.2 Input Buffer Format

Algorithm expects the features to be 16 bit data. The details about the feature planes regarding, number of scales, feature columns, feature rows, feature pitch, number of planes etc must be written at the beginning of the buffer. This data structure must comply to TI_OD_featMetaData specified in iobjdet.h interface file.

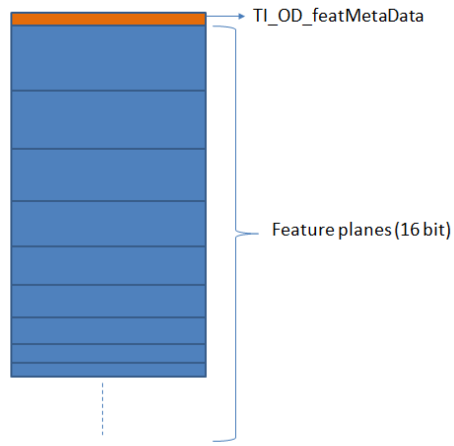

Figure 3 Input Buffer format

4.6.3 Output Buffer Format

The object detection module outputs the number of objects detected via TI_OD_output structure defined in iobjdet.h interface. The structure provides the number of objects detected and also the list of objects detected. Please refer to section 5.1.11.11 for more details.

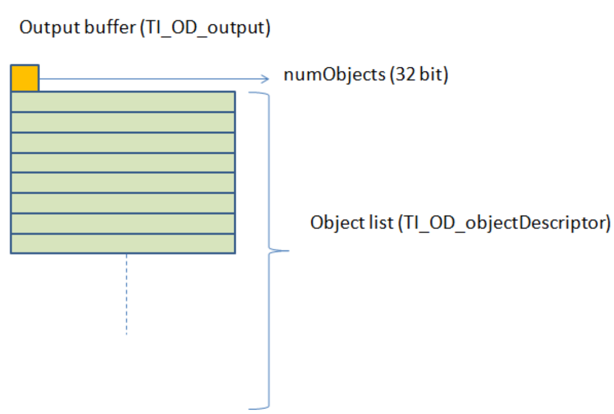

Figure 4 Output Buffer format

5 API Reference

This chapter provides a detailed description of the data structures and interfaces functions used by Object Detection.

5.1.1 IVISION_Params

Description

This structure defines the basic creation parameters for all vision applications.

Fields

Field	Data Type	Input/ Output	Description
algParams	IALG_Params	Input	IALG Params
cacheWriteBack	ivisionCacheWriteBack	Input	Function pointer for cache write back for cached based system. If the system is not using cache fordata memory then the pointer can be filled with NULL. If the algorithm recives a input buffer with IVISION_AccessMode as IVISION_ACCESSMODE_CPU and the ivisionCacheWriteBack as NULL then the algorithm will return with error

5.1.2 IVISION_Point

Description

This structure defines a 2-dimensional point

Fields

Field	Data Type	Input/ Output	Description
X	XDAS_Int32	Input	X (horizontal direction offset)
y	XDAS_Int32	Input	Y (vertical direction offset)

5.1.3 IVISION_Rect

Description

This structure defines a rectangle

Fields

Field	Data Type	Input/ Output	Description
`topLeft`	`XDAS_Int32`	Input	Top left co-ordinate of rectangle
`width`	`XDAS_Int32`	Input	Width of the rectangle
`height`	`XDAS_Int32`	Input	Height of the rectangle

5.1.4 IVISION_Polygon

Description

This structure defines a poylgon

Fields

Field	Data Type	Input/ Output	Description
`numPoints`	`XDAS_Int32`	Input	Number of points in the polygon
`poits`	`IVISION_Point*`	Input	Points of polygon

5.1.5 IVISION_BufPlanes

Description

This structure defines a generic plane descriptor

Fields

Field	Data Type	Input/ Output	Description
`buf`	`Void*`	Input	Number of points in the polygon
`width`	`XDAS_UInt32`	Input	Width of the buffer (in bytes), This field can be viewed as pitch while processing a ROI in the buffer
`height`	`XDAS_UInt32`	Input	Height of the buffer (in lines)

Field	Data Type	Input/ Output	Description
frameROI	IVISION_Rect	Input	Region of the intererst for the current frame to be processed in the buffer. Dimensions need to be a multiple of internal block dimenstions. Refer application specific details for block dimensions supported for the algorithm. This needs to be filled even if bit-0 of IVISION_InArgs::subFrameInfo is set to 1
subFrameROI	IVISION_Rect	Input	Region of the intererst for the current sub frame to be processed in the buffer. Dimensions need to be a multiple of internal block dimenstions. Refer application specific details for block dimensions supported for the application. This needs to be filled only if bit-0 of IVISION_InArgs::subFrameInfo is set to 1
freeSubFrameROI	IVISION_Rect	Input	This ROI is portion of subFrameROI that can be freed after current slice process call. This field will be filled by the algorithm at end of each slice processing for all the input buffers (for all the output buffers this field needs to be ignored). This will be filled only if bit-0 of IVISION_InArgs::subFrameInfois set to 1
planeType	XDAS_Int32	Input	Content of the buffer - for example Y component of NV12
accessMask	XDAS_Int32	Input	Indicates how the buffer was filled by the producer, It is IVISION_ACCESSMODE_HWA or IVISION_ACCESSMODE_CPU

5.1.6 IVISION_BufDesc

Description

This structure defines the iVISION buffer descriptor

Fields

Field	Data Type	Input/ Output	Description
numPlanes	Void*	Input	Number of points in the polygon
bufPlanes[IVISION_MAX_NUM_PLANES]	IVISION_BufPlanes	Input	Description of each plane
formatType	XDAS_UInt32	Input	Height of the buffer (in lines)

Field	Data Type	Input/ Output	Description
bufferId	XDAS_Int32	Input	Identifier to be attached with the input frames to be processed. It is useful when algorithm requires buffering for input buffers. Zero is not supported buffer id and a reserved value
Reserved[2]	XDAS_UInt32	Input	Reserved for later use

5.1.7 IVISION_BufDescList

Description

This structure defines the iVISION buffer descriptor list. IVISION_InBufs and IVISION_OutBufs is of the same type

Fields

Field	Data Type	Input/ Output	Description
Size	XDAS_UInt32	Input	Size of the structure
numBufs	XDAS_UInt32	Input	Number of elements of type IVISION_BufDesc in the list
bufDesc	IVISION_BufDesc **	Input	Pointer to the list of buffer descriptor

5.1.8 IVISION_InArgs

Description

This structure defines the iVISION input arugments

Fields

Field	Data Type	Input/ Output	Description
Size	XDAS_UInt32	Input	Size of the structure
subFrameInfo	XDAS_UInt32	Input	bit0 - Sub frame processing enable (1) or disabled (0) bit1 - First subframe of the picture (0/1) bit 2 - Last subframe of the picture (0/1) bit 3 to 31 – reserved

5.1.9 IVISION_OutArgs

Description

This structure defines the iVISION output arugments

Fields

Field	Data Type	Input/ Output	Description
`Size`	`XDAS_UInt32`	Input	Size of the structure
`inFreeBufIDs[IVISION_MAX_NUM_FREE_BUFFERS]`	`XDAS_UInt32`	Input	Array of bufferId's corresponding to the input buffers that have been unlocked in the Current process call. The input buffers released by the algorithm are indicated by their non-zero ID (previously provided via IVISION_BufDesc#bufferId A value of zero (0) indicates an invalid ID. The first zero entry in array will indicate end of valid inFreeBufIDs within the array hence the application can stop searching the array when it encounters the first zero entry. If no input buffer was unlocked in the process call, inFreeBufIDs[0] will have a value of zero.
`outFreeBufIDs [IVISION_MAX_NUM_FREE_BUFFERS]`	`XDAS_UInt32`	Input	Array of bufferId's corresponding to the Output buffers that have been unlocked in the Current process call. The output buffers released by the algorithm are indicated by their non-zero ID (previously provided via IVISION_BufDesc#bufferId A value of zero (0) indicates an invalid ID. The first zero entry in array will indicate end of valid inFreeBufIDs within the array hence the application can stop searching the array when it encounters the first zero entry. If no output buffer was unlocked in the process call, inFreeBufIDs[0] will have a value of zero.
`reserved[2]`	`XDAS_UInt32`		Reserved for future usage

5.1.10 Object Detection Enumeration

This section includes the following Object Detection specific enumerations:

- TI_OD_ObjectType
- TI_OD_InBufOrder
- TI_OD_OutBufOrder
- TI_OD_ROIPreset
- TI_OD_TSRClassTemplates

5.1.10.1 TI_ OD_ObjectType

Description

Enum to indicate type of object detected. This is used to populate objType in TI_OD_output structure

Fields

Field	Value	Description
TI_OD_PEDESTRIAN	0	Indicates that the detected object type is Pedestrian
TI_OD_TRAFFIC_SIGN	1	Indicates that the detected object type is Traffic sign
TI_OD_MAX_OBJECTS	2	Maximum number of objects supported

5.1.10.2 TI_ OD_InBufOrder

Description

User provides most of the infomration through buffer descriptor during process call. Below enums define the purpose of input buffer.There is 1 input buffer descriptor

Fields

Field	Value	Description
TI_OD_IN_BUFDESC_FEATURE_PLANES	0	This buffer descriptor provides the feature planes representing the image. The feature planes are assumed to be 16 bit data.
TI_OD_IN_BUFDESC_TOTAL	1	Total number of input buffer descriptor

5.1.10.3 TI_ OD_OutBufOrder

Description

User provides most of the infomration through buffer descriptor during process call. Below enums define the purpose of output buffer.There is 1 output buffer descriptor

Fields

Field	Value	Description
TI_OD_OUT_BUFDESC_LIST	0	This buffer is filled up by algorithm with a list of detected objects.
TI_OD_OUT_BUFDESC_TOTAL	1	Total number of output buffer descriptor

5.1.10.4 TI_ OD_ROIPreset

Description

ROI processing presets supported by OD module.

Fields

Field	Value	Description
TI_OD_ROI_FULL_FRAME	0	This preset will enable search at every point in the provided feature data
TI_OD_ROI_DYNAMIC	1	This preset will enable search only a small region around a detected object in the previous frame. The previous frame could have been fully processed or processed based on detections.

5.1.10.5 TI_ OD_TSRClassTemplates

Description

Enumeration of German Traffic signs. The current verion of OD supports only the below listed 43 clases

Fields

Field	Value	Description
TI_OD_TSR_SPEED_LIMIT_20	0	Enumeration for "speed limit 20"
TI_OD_TSR_SPEED_LIMIT_30	1	Enumeration for "speed limit 30"
TI_OD_TSR_SPEED_LIMIT_50	2	Enumeration for "speed limit 50"

Field	Value	Description
TI_OD_TSR_SPEED_LIMIT_60	3	Enumeration for "speed limit 60"
TI_OD_TSR_SPEED_LIMIT_70	4	Enumeration for "speed limit 70"
TI_OD_TSR_SPEED_LIMIT_80	5	Enumeration for "speed limit 80"
TI_OD_TSR_RESTRICTION_ENDS_80	6	Enumeration for "restriction ends 80"
TI_OD_TSR_SPEED_LIMIT_100	7	Enumeration for "speed limit 100"
TI_OD_TSR_SPEED_LIMIT_120	8	Enumeration for "speed limit 120"
TI_OD_TSR_NO_OVERTAKING	9	Enumeration for "no overtaking"
TI_OD_TSR_NO_OVERTAKING_TRUCKS	10	Enumeration for "no overtaking (trucks)"
TI_OD_TSR_PRIORITY_AT_NEXT_INTERSECTION	11	Enumeration for "priority at next intersection"
TI_OD_TSR_PRIORITY_ROAD	12	Enumeration for "priority road"
TI_OD_TSR_GIVE_WAY	13	Enumeration for "give way"
TI_OD_TSR_STOP	14	Enumeration for "stop"
TI_OD_TSR_NO_VEHICLES	15	Enumeration for "no vehicles"
TI_OD_TSR_NO_TRUCKS	16	Enumeration for "no trucks"
TI_OD_TSR_NO_ENTRY	17	Enumeration for "no entry"
TI_OD_TSR_DANGER_AHEAD	18	Enumeration for "danger ahead"
TI_OD_TSR_BEND_TO_LEFT	19	Enumeration for "bend to left"
TI_OD_TSR_BEND_TO_RIGHT	20	Enumeration for "bend to right"
TI_OD_TSR_DOUBLE_BEND_FIRST_TO_LEFT	21	Enumeration for "double bend (first to left)"
TI_OD_TSR_UNEVEN_ROAD	22	Enumeration for "uneven road"
TI_OD_TSR_SLIPPERY_ROAD	23	Enumeration for "slippery road"
TI_OD_TSR_ROAD_NARROWS	24	Enumeration for "road narrows"
TI_OD_TSR_CONSTRUCTION	25	Enumeration for "construction"
TI_OD_TSR_TRAFFIC_SIGNAL	26	Enumeration for "traffic signal"
TI_OD_TSR_PEDESTRIAN_CROSSING	27	Enumeration for "pedestrian crossing"

Field	Value	Description
`TI_OD_TSR_SCHOOL_CROSSING`	28	Enumeration for "school crossing"
`TI_OD_TSR_CYCLES_CROSSING`	29	Enumeration for "cycles crossing"
`TI_OD_TSR_SNOW`	30	Enumeration for "snow"
`TI_OD_TSR_ANIMALS`	31	Enumeration for "animals"
`TI_OD_TSR_RESTRICTION_ENDS`	32	Enumeration for "restriction ends"
`TI_OD_TSR_GO_RIGHT`	33	Enumeration for "go right"
`TI_OD_TSR_GO_LEFT`	34	Enumeration for "go left"
`TI_OD_TSR_GO_STRAIGHT`	35	Enumeration for "go straight"
`TI_OD_TSR_GO_RIGHT_OR_STRAIGHT`	36	Enumeration for "go right or straight"
`TI_OD_TSR_GO_LEFT_OR_STRAIGHT`	37	Enumeration for "go left or straight"
`TI_OD_TSR_KEEP_RIGHT`	38	Enumeration for "keep right"
`TI_OD_TSR_KEEP_LEFT`	39	Enumeration for "keep left"
`TI_OD_TSR_ROUNDABOUT`	40	Enumeration for "roundabout"
`TI_OD_TSR_RESTRICTION_ENDS_OVERTAKING`	41	Enumeration for "restriction ends (overtaking)"
`TI_OD_TSR_RESTRICTION_ENDS_OVERTAKING_FOR_TRUCKS`	42	Enumeration for "restriction ends (overtaking (trucks))"
`TI_OD_TSR_NOT_A_TRAFFIC_SIGN`	43	Enumeration for "Not a traffic sign"

5.1.11 Object Detection Data Structures

This section includes the following Object Detection specific extended data structures:

- TI_OBJECT_FEATURES_scaleMetaData
- TI_ OBJECT_FEATURES_outputMetaData
- TI_ OBJECT_FEATURES_CreateParams
- TI_ OBJECT_FEATURES_PDConfig
- TI_ OBJECT_FEATURES_TSRConfig
- TI_ OBJECT_FEATURES_InArgs
- TI_ OBJECT_FEATURES_PDStats
- TI_ OBJECT_FEATURES_TSRStats
- TI_ OBJECT_FEATURES_OutArgs
- TI_ OBJECT_FEATURES_objectDescriptor
- TI_ OBJECT_FEATURES_output

5.1.11.1 TI_ OBJECT_FEATURES_scaleMetaData

Description

This structure defines the scale parameters of the input feature plane. This is a common data structure/interface agreed upon between feature generation module and object detection module. This information is assumed to be written by the feature generation module at the beginning of each feature plane input buffer.

Fields

Field	Data Type	Input/ Output	Description
scaleOffset	uint32_t	Input	Byte offset from the beginning of input buffer
orgImCols	uint16_t	Input	Original width of the image
orgImRows	uint16_t	Input	Original height of the image
imCols	uint16_t	Input	ROI width of the image
imRows	uint16_t	Input	ROI height of the image
startX	uint16_t	Input	Starting X location of the ROI in the original image.

Field	Data Type	Input/ Output	Description
startY	uint16_t	Input	Starting Y location of the ROI in the original image.
featCols	uint16_t	Input	Width of the feature plane
featRows	uint16_t	Input	Height of the feature plane
featPitch	uint16_t	Input	Pitch of the feature plane
planeOffset	uint32_t	Input	Offset between feature planes

5.1.11.2 TI_ OBJECT_FEATURES_outputMetaData

|| Description

This structure contains the meta data generated by the feature generation module. The feature generation module is responsible of populating this structure. The format of this structure agreed upon by feature generation module and object detection module.

|| Fields

Field	Data Type	Input/ Output	Description
size	uint32_t	Input	Size of TI_OD_featMetaData structure
featBufSize	uint32_t	Input	Total size of input buffer which includes size of feature planes and TI_OD_featMetaData structure
numScales	uint8_t	Input	Number of feature scales
numPlanes	uint8_t	Input	Number of feature planes
outFormat	uint8_t	Input	Format of feature planes, interleaved/deinterleaved
leftPadPels	uint16_t	Input	Amount of padded pixels from left of image boundary

Field	Data Type	Input/ Output	Description
topPadPels	uint16_t	Input	Amount of padded pixels from the top of image boundary
computeCellSum	uint8_t	Input	Flag to indicate if 2x2 cell sum is to be computed by object detection or not. 0 – Cell sum computed externally and not required for this module to compute 1 – Cell sum to be computed by object detection module
scaleInfo[TI_OD_MAX_TOTAL_SCALES]	TI_OD_scaleMetaData	Input	List of feature scale metadata as defined by TI_OD_scaleMetaData

5.1.11.3 TI_OBJECT_FEATURES_CreateParams

|| Description

This structure defines the run-time input arguments for Object Detection instance object.

|| Fields

Field	Data Type	Input/ Output	Description
visionParams	IVISION_Params	Input	See IVISION_Params data structure for details
edma3RmLldHandle	void *	Input	Pointer to edma3-lld resource manager
maxImageWidth	uint16_t	Input	Max input width of image
maxImageHeight	uint16_t	Input	Max input height of image
maxScales	uint16_t	Input	Max number of supported scales

5.1.11.4 TI_OD_PDConfig

|| Description

This structure contains the PD specific config parameters.

|| Fields

Field	Data Type	Input/ Output	Description
enablePD	uint8_t	Input	Flag to enable or disable pedestrian detection 0 - disable 1 - enable (default)
classifierTypePD	uint8_t	Input	Flag to indicate type of classifier to be used 0 - 2 level Adaboost (default)
trackingMethodPD	uint8_t	Input	Flag to enable / disable pedestrian tracking 0 - disable 1 - enable Kalman filter based tracking (default)
softCascadeThPD	int32_t	Input	32-bit signed threshold value for AdaBoost -1 (recommended)
strongCascadeThPD	int32_t	Input	32-bit signed threshold value for AdaBoost -1 (recommended)

5.1.11.5 TI_OD_TSRConfig

|| Description

This structure contains the TSR specific config parameters.

|| Fields

Field	Data Type	Input/ Output	Description
enableTSR	uint8_t	Input	Flag to enable or disable traffic sign detection 0 - disable 1 - enable (default)
classifierTypeTSR	uint8_t	Input	Flag to indicate type of classifier to be used 0 - 2 level Adaboost (default)
trackingMethodTSR	uint8_t	Input	Flag to enable / disable pedestrian tracking 0 – disable (default) 1 - enable Kalman filter based tracking

Field	Data Type	Input/ Output	Description
recognitionMethodTSR	uint8_t	Input	Flag to enable / disable traffic sign recognition. 0 - disable 1 - enable kalman filter based tracking (default)
softCascadeThTSR	int32_t	Input	16-bit signed threshold value for AdaBoost -1 (recommended)
strongCascadeThTSR	int32_t	Input	16-bit signed threshold value for AdaBoost 7 (recommended)

5.1.11.6 TI_OD_InArgs

|| Description

This structure contains all the parameters which are given as input to OD algorithm at frame level

|| Fields

Field	Data Type	Input/ Output	Description
iVisionInArgs	IVISION_InArgs	Input	See IVISION_InArgs data structure for details.
pdConfig	TI_OD_PDConfig	Input	See TI_OD_PDConfig data structure for details.
tsrConfig	TI_OD_TSRConfig	Input	See TI_OD_TSRConfig data structure for details.
detectionMode	uint8_t	Input	This is a performance knob to control search points in feature plane When, detectionMode = 0 => Search all points. HIGH_QUALITY mode. (default) detectionMode = 1 => Skip odd points in the horizontal direction
roiPreset	uint8_t	Input	This flag enables or disables dynamic ROI mode. 0 – Disable dynamic ROI 1 – Enable dynamic ROI

Field	Data Type	Input/ Output	Description
refreshInterval	uint8_t	Input	Valid only when roiPreset = 1, (default) A value of 0 will enable full frame processing for all frames (F, F, F, F, ...) A value of 1 will enable full frame processing for every other frame (F, R, F, R, ...) A value of 2 will enable full frame processing after two frames (F, R, R, F, R, R, ...) And so on. Max value is 10.
reserved0	uint32_t	Input	Reserved 32-bit field. Must be set to 0 for normal operation
reserved1	uint32_t	Input	Reserved 32-bit field. Must be set to 0 for normal operation
reserved2	uint32_t	Input	Reserved 32-bit field. Must be set to 0 for normal operation
reserved3	uint32_t	Input	Reserved 32-bit field. Must be set to 0 for normal operation

5.1.11.7 TI_OD_PDStats

|| Description

This structure reports PD statistics, to be used only for debugging.

|| Fields

Field	Data Type	Input/ Output	Description
numCyclesPD	uint32_t	Output	Number of cycles taken by classifier
numTreesPD	uint32_t	Output	Total number of trees traversed by AdaBoost classifier NA when classifierType is not AdaBoost.

5.1.11.8 TI_OD_TSRStats

|| Description

This structure reports TSR statistics, to be used only for debugging.

|| Fields

Field	Data Type	Input/ Output	Description
numCyclesTSR	uint32_t	Output	Number of cycles taken by classifier
numTreesTSR	uint32_t	Output	Total number of trees traversed by AdaBoost classifier. NA when classifierType is not AdaBoost.

5.1.11.9 TI_OD_OutArgs

|| Description

This structure contains all the parameters which are given as output by the algorithm.

|| Fields

Field	Data Type	Input/ Output	Description
iVisionOutArgs	IVISION_OutArgs	Output	See IVISION_OutArgs data structure for details.
pdStats	TI_OD_PDStats	Output	See TI_OD_PDStats data structure for details.
tsrStats	TI_OD_TSRStats	Output	See TI_OD_TSRStats data structure for details.

5.1.11.10 TI_OD_objectDescriptor

|| Description

This structure contains the detected object properties such as location-(x, y), size-(height, width), confidence (score) type - (objTag), string messages etc.

|| Fields

Field	Data Type	Input/ Output	Description
objType	uint8_t	Output	See TI_OD_ObjectType enum for details.
xPos	uint16_t	Output	Location of the detected object in the image along X direction
yPos	uint16_t	Output	Location of the detected object in the image along Y direction

Field	Data Type	Input/ Output	Description
`objWidth`	`uint16_t`	Output	Width of the located object in pixels. Does not indicate actual width of the object.
`objHeight`	`uint16_t`	Output	Width of the located object in pixels. Does not indicate actual height of the object.
`objTag`	`uint32_t`	Output	Value or Index to indicate, color in case of pedestrian tracking or traffic sign meaning. This field can be used to pass an index to a color array or definition array for PD/TSR etc. Eg. for TSR this field will be populated with one of the enumeration type defined by TI_OD_TSRClassTemplates, indicating the type of traffic sign. For PD, it is don't care in this release
`objScore`	`int32_t`	Output	Confidence measure of detected object
`numMsg`	`uint16_t`	Output	Number of auxiliary string messages passed by algorithm back to the application. Max is defined by MAX_NUM_OUPUT_STRINGS
`objMsg [MAX_NUM_OUPUT_STRINGS][MAX_STRING_SIZE]`	`uint8_t`	Output	Auxiliary string message describing the object by the algorithm. Only for display purpose.

5.1.11.11 TI_OD_output

‖ Description

This is the output structure given out by object detection module. It contains the number of objects detected and TI_OD_MAX_DETECTIONS_PER_FRAME instances of TI_OD_objectDescriptor structure. The number of valid descriptors is governed by numObjects variable.

‖ Fields

Field	Data Type	Input/ Output	Description
`numObjects`	`int32_t`	Output	Number of objects detected by the module
`objDesc[TI_OD_MAX_DETECTIONS_PER_FRAME]`	`TI_OD_objectDescriptor`		See `TI_OD_objectDescriptor` for more details

5.2 Default and Supported Values of Parameter

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

- `TI_OD_PDConfig`
- `TI_OD_TSRConfig`

Table 4 Default and Supported Values for TI_OD_PDConfig

Field	Default Value	Supported Value
enablePD	1	❑ 0 – disable PD ❑ 1 – enable PD
classifierTypePD	0	❑ 0 – AdaBoost ❑ 1 – reserved
trackingMethodPD	1	❑ 0 – disable tracking ❑ 1 – kalman filter based tracking
softCascadeThPD	-1	A value of -1 indicates -8192 and +1 indicates +8192 in Q-13 format. Setting a value of -1 will output all detected windows.
strongCascadeThPD	-1	A value of -1 indicates -8192 and +1 indicates +8192 in Q-13 format. Setting a value of -1 will output all detected windows.

Table 5 Default and Supported Values for TI_OD_TSRConfig

Field	Default Value	Supported Value
enableTSR	1	❑ 0 – disable TSR ❑ 1 – enable TSR
classifierTypeTSR	0	❑ 0 – AdaBoost ❑ 1 – reserved
trackingMethodTSR	0	❑ 0 – disable tracking ❑ 1 – kalman filter based tracking
recognitionMethodTSR	1	❑ 0 – disable recognition ❑ 1 – enable recognition
softCascadeThTSR	-1	A value of -1 indicates -8192 and +1 indicates +8192 in Q-13 format. Setting a value of -1 will output

Field	Default Value	Supported Value
		all detected windows.
strongCascadeThTSR	7	A value of -1 indicates -8192 and +1 indicates +8192 in Q-13 format. Setting a value of -1 will output all detected windows.

5.3 Interface Functions

This section describes the Application Programming Interfaces (APIs) used by Object detection. 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).

5.4 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()

5.5 Initialization API

Initialization API is used to initialize an instance of the algorithm. The initialization parameters are defined in the `IVISION_Params` structure (see section 5.1.1 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()
```

5.6 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 `IALG_Cmd` data structure.

‖ Name

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

‖ Synopsis

XDAS_Int32 (*control) (IVISION_Handle handle, IALG_Cmd id, IALG_Params *inParams, IALG_Params *outParams);

‖ Arguments

IVISION_Handle handle; /* algorithm instance handle */

IALG_Cmd id; /* algorithm specific control commands*/

IALG_Params *inParams /* algorithm input parameters */

IALG_Params *outParams /* algorithm output parameters */

‖ Return Value

IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */

‖ Description

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

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

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

‖ 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 `IALG_Params` structure.
-

‖ 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 Object Detection 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 `IALG_EOK`.
- The algorithm should not modify the contents of params. That is, the data pointed to by this parameter must be treated as read-only.

‖ Example

See test bench file, object_detection_tb.c available in the \test\src sub-directory.

‖ See Also

algInit(), algActivate(), process()

5.7 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)(IVISION_Handle handle, IVISION_inBufs *inBufs, IVISION_outBufs *outBufs, IVISION_InArgs *inargs, IVISION_OutArgs *outargs);

|| Arguments

IVISION_Handle handle; /* algorithm instance handle */

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

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

IVISION_InArgs *inargs /* algorithm runtime input arguments */

IVISION_OutArgs *outargs /* algorithm runtime output arguments */

|| Return Value

IALG_EOK; /* status indicating success */

IALG_EFAIL; /* status indicating failure */

|| Description

This function does the basic object detection. 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 `IVISION_inBufs, IVISION_outBufs` data structure for details).

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

The last argument is a pointer to the `IVISION_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()`.
- 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 `IVISION_InArgs` structure.
- `outArgs` must not be NULL and must point to a valid `IVISION_OutArgs` structure.
- `inBufs` must not be NULL and must point to a valid `IVISION_inBufs` 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 `IVISION_outBufs` 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.
- 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 `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 `#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, object_detection_tb.c available in the \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.

|| 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()`

5.8 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