Circular Light Recognition using TI's TMS320C66x DSP

User Guide

June 2017

Products

Audio	www.ti.com/audio
Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
OMAP Applications Processors	www.ti.com/omap
Wireless Connectivity	www.ti.com/wirelessconnectivity

Applications

Automotive & Transportation	www.ti.com/automotive
Communications & Telecom	www.ti.com/communications
Computers & Peripherals	www.ti.com/computers
Consumer Electronics	www.ti.com/consumer-apps
Energy and Lighting	www.ti.com/energyapps
Industrial	www.ti.com/industrial
Medical	www.ti.com/medical
Security	www.ti.com/security
Space, Avionics & Defense	www.ti.com/space-avionics-defense
Video & Imaging	www.ti.com/video
TI E2E Community	e2e.ti.com

1 Read This First

1.1 About This Manual

This document describes how to install and work with Texas Instruments' (TI) Circular Light Recognition 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 Circular Light Recognition 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 Circular Light Recognition 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 Circular Light Recognition 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
DMA	Direct Memory Access
DSP	Digital Signal Processing
EVM	Evaluation Module
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 (Circular Light Recognition Module on TMS320C66x DSP) and version number. The version number of the Circular Light Recognition Module is included in the Title of the Release Notes that accompanies the product release.

1.8 Trademarks

Code Composer Studio, eXpressDSP, Circular Light Recognition 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 Circular Light Recognition 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 Circular Light Recognition

The Circular Light Recognition module can be used to detect and recognize the color of circular lights such as traffic lights and vehicle lights. This algorithm uses color segmentation and Hough circle transform as basic modules. The below block diagram shows the building blocks of this algorithm with their data flows.

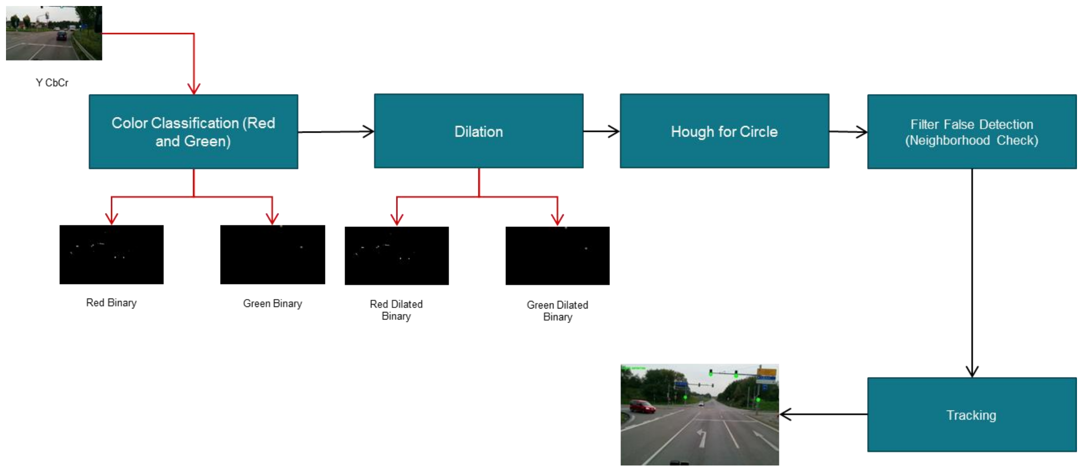


Figure 1 Fundamental blocks of Circular Light Recognition

2.3 Supported Services and Features

This user guide accompanies TI's implementation of Circular Light Recognition Algorithm on the TI's C66x DSP.

This version of the Circular Light Recognition modules can be tuned to recognize traffic lights and circular head lights of vehicles

3 Installation Overview

This chapter provides a brief description on the system requirements and instructions for installing Circular Light Recognition 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 Circular Light Recognition 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.CLR.C66x.00.02. Folder structure of this top level directory is shown in below table.

Table 2 Component Directories in case of Object release

Sub-Directory	Description

Sub-Directory	Description
\common	Common files for building different modules
\makerules	Make rules files
\modules	Top level folder containing different DSP app modules
\modules \ti_circular_light_recognition	Circular Light Recognition module for C66x DSP
\modules \ti_circular_light_recognition \docs	User guide and Datasheet for Circular Light Recognition module
\modules \ti_circular_light_recognition \inc	Contains iclr_ti.h interface file
\modules \ti_circular_light_recognition \lib	Contains Circular Light Recognition algorithm library
\modules \ti_circular_light_recognition \test	Contains standalone test application source files
\modules \ti_circular_light_recognition \test\out	Contains test application .out executable
\modules \ti_circular_light_recognition \test\src	Contains test application source files
\modules \ti_circular_light_recognition \test\testvecs	Contains config, input, output, reference test vectors
\modules \ti_circular_light_recognition \test\testvecs\config	Contain config file to set various parameters exposed by Circular Light Recognition module
\modules \ti_circular_light_recognition \test\testvecs\input	Contains sample input feature vector .bin file

Sub-Directory	Description
`\modules \ti_circular_light_recognition \test\testvecs\output`	`Contains output .txt file with a list of objects detected`
`\modules \ti_circular_light_recognition \test\testvecs\reference`	`Contains reference .txt file with a list of objects detected`

3.3 Building Sample Test Application

This Circular Light Recognition library has been accompanied by a sample test application. To run the sample test application XDAIS tools are required.

This version of the Circular Light Recognition library has been validated with XDAIS tools containing IVISION interface version. Please refer to release notes for dependent components and their version.

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 C66x VLIB

Install C66x VLIB version 3.3.0.1 from the link

http://software-dl.ti.com/libs/vlib/latest/index_FDS.html

After installing VLIB, set the environment variable to “`VLIB_INSTALL_DIR`” to the installed directory like <install directory>\packages

3.3.4 DMA utility Library

Install DMA utility library for DSP from the link

https://cdds.ext.ti.com/ematrix/common/emxNavigator.jsp?objectId=28670.42872.62652.37497

After installing the DMA Utility Library, Set a system environment variable named "DMAUTILS_PATH" pointing to <install directory>\ dmautils

Note: This DMAUTILS component is also available as part of Processor SDK Package.

3.3.5 Building the Test Application Executable through GMAKE

The sample test application that accompanies Circular Light Recognition 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_circular_light_recognition\test> gmake clean

    b) modules\ti_circular_light_recognition\test> gmake all

5) The above step creates an executable file, ti_circular_light_recognition_algo.out in the modules\ti_circular_light_recognition\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_circular_light_recognition\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 Circular Light Recognition 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 (clr.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, clr.cfg contains the configuration parameters required for the algorithm. The clr.cfg file is available in the \test\testvecs\config sub-directory.

A sample clr.cfg file is as shown.

```
#-------------------------------------------------------------------#
# Common Parameters                                                 #
#-------------------------------------------------------------------#
numTestCases     = 1

inFileNameYUV    = "../testvecs/input/country_night1_1280x720.yuv"

outFileNameYUV   = "../testvecs/output/country_night1_1280x720_out.yuv"

traceNameYUV     = "../testvecs/output/country_night1_1280x720_trace.yuv"

outFileName      = "../testvecs/output/country_night1_1280x720.txt"

maxImageWidth    = 1280

maxImageHeight   = 720

actualImgWidth   = 1280

actualImgHeight  = 720

startFrame       = 0

maxFrames        = 1000

roiWidth         = 600

roiHeight        = 384

startX           = 8

startY           = 300

maxNumRadius     = 8
```

```
numRadius        = 6

trackingMethod   = 1

groupingWindowSize = 60

morphologyMethod   = 1

lightSelection     = 1

useCacheForList    = 0

lightColor         = 0 1

lightBrightnessThr = 180 220

lightThr1          = 115 113

lightThr2          = 140 143

falseFilterThr     = 100000 600

circleDetectionThr = 120 120 120 120 120

radius             = 6 7 9 12 16 20

scalingFactor      = 0 1 1 2 2 2
```

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

3.5 Host emulation build for source package

The source release of Circular Light Recognition module can be built in host emulation mode. This option speeds up development and validation time by running the platform code on x86/x64 PC.

3.5.1 Installing Visual Studio

Building host emulation for Circular Light Recognition module requires Microsoft Visual Studio 11.0 (2012) which can be downloaded from below link.

http://www.microsoft.com/en-in/download/details.aspx?id=34673

3.5.2 Installing VLIB package for host emulation

Circular Light Recognition source package relies on VLIB source package to build the target in host emulation mode. Install VLIB package and link the pre-built host emulation VLIB libraries against Circular Light Recognition module.

After installing VLIB, set the environment variable to “`VLIB_HOST_INSTALL_DIR`” to the installed directory like <install directory>\packages

3.5.3 Building source in host emulation

After installing the required components, navigate to Circular Light Recognition install path and run `vcvarsall.bat` to setup the required environment variables

```
{clr_install_path} > {…\Microsoft Visual Studio
11.0\VC\vcvarsall.bat}
```

Once the environment variables are setup build the Circular Light Recognition source in host emulation mode

```
{clr_install_path} > gmake all TARGET_BUILD=debug
TARGET_PLATFORM=PC
```

This will build the host emulation executable under the path

```
{clr_install_path}\test\out\ti_circular_light_recognition_algo.out.
exe
```

To build the example in host emulation mode for c6x DSP you will need to install the c6xsim which is available at
http://processors.wiki.ti.com/index.php/Run_Intrinsics_Code_Anywhere

Install this package in the common folder

3.5.4 Running host emulation executable

Launch Microsoft Visual Studio 11.0 and open file
`ti_circular_light_recognition_algo.out.exe`

This will load the host emulation program which can be used for development and validation purpose.

3.6 Uninstalling the Component

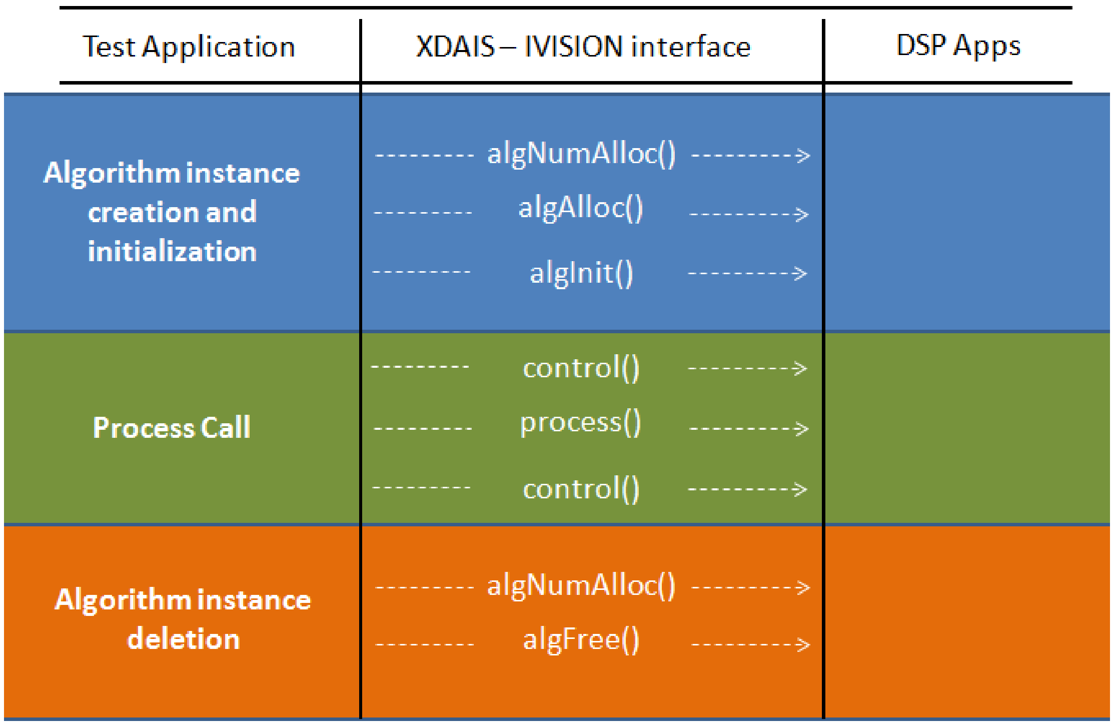
To uninstall the component, delete the algorithm directory from your hard disk.

4 Sample Usage

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

4.1 *Overview of the Test Application*

The test application exercises the `IVISION` and extended class of the Circular Light Recognition 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, Circular Light Recognition 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 clr.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_CLR_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.

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 Output Buffer Format

The Circular Light Recognition module outputs the number of objects detected via TI_CLR_output structure defined in iclr_ti.h interface. The structure provides the number of lights detected and also the list of lights detected. Please refer to section 5.1.11.5 for more details.

5 API Reference

This chapter provides a detailed description of the data structures and interfaces functions used by Circular Light Recognition.

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 Circular Light Recognition Enumeration

This section includes the following Circular Light Recognition specific enumerations:

`TI_CLR_HoughSpaceScaling`

`TI_CLR_HoughSpaceVotingMethod`

`TI_CLR_TrackingMethod`

`TI_CLR_InBufOrder`

`TI_CLR_OutBufOrder`

`TI_CLR_PrimaryColor`

`TI_CLR_LightSelectionMethod`

`TI_CLR_ObjectType`

`TI_CLR_ErrorType`

5.1.10.1 TI_CLR_HoughSpaceScaling

Description

Enum to indicate type of downscaling to be performed in the output Hough space during Circle detection

Fields

Field	Description
`TI_CLR_HS_NO_SCALING`	No averaging in Hough space
`TI_CLR_HS_SCALE_2x2`	Averaging on a 2x2 grid.
`TI_CLR_HS_SCALE_4x4`	Averaging on a 4x4 grid.

5.1.10.2 TI_CLR_HoughSpaceVotingMethod

Description

Enum to indicate type of Voting method for Hough circle transformFields

Field	Description
`TI_CLR_HSV_DEFAULT`	TI_CLR_HSV_GRAD_MAGNITUDE

Field	Description
`TI_CLR_HSV_GRAD_MAGNITUDE`	Increment by gradient magnitude.
`TI_CLR_HS_MAX`	Max value

5.1.10.3 TI_CLR_TrackingMethod

Description

Enum to indicate List of supported tracking method

Field	Description
`TI_CLR_TRACKING_NONE`	No tracking/tracking disables
`TI_CLR_TRACKING_KALMAN`	Kalman filter based tracking
`TI_CLR_TRACKING_MAX`	Max value

5.1.10.4 TI_CLR_InBufOrder

Description

User provides most of the information through buffer descriptor during process call. Below enum define the purpose of input buffer. There is only one input buffer descriptor

Field	Description
`TI_CLR_BUFDESC_IN_IMAGEBUFFER`	This buffer descriptor provides the actual image data required by algorithm
`TI_CLR_BUFDESC_IN_TOTAL`	Total number of in buffers

5.1.10.5 TI_CLR_OutBufOrder

Description

User provides most of the information through buffer descriptor during process call. Below enum define the purpose of output buffer. There is only one output buffer descriptor

Field	Description

Field	Description
TI_CLR_BUFDESC_OUT_OBJ_BUFFER	This buffer descriptor has the details/properties of all the detected circular lights
TI_CLR_BUFDESC_OUT_TOTAL	Total number of out buffers

5.1.10.6 TI_CLR_PrimaryColor

Description

This enum indicates the List of Primary color types supported.

Field	Description
TI_CLR_PC_RED	RED light
TI_CLR_PC_GREEN	GREEN Light
TI_CLR_PC_MAX	Max value

5.1.10.7 TI_CLR_LightSelectionMethod

Description

This enum indicates the List of Light selection methods suported.

Field	Description
TI_CLR_PC_RED	RED light
TI_CLR_LSM_DEFAULT	TI_CLR_LSM_RED_GREEN
TI_CLR_LSM_RED_GREEN	RED and GREEN Light
TI_CLR_LSM_CUSTOM	lightBrightnessThr, lightColor etc will be used for
TI_CLR_LSM_MAX	Max value

5.1.10.8 TI_CLR_ErrorType

Description

Enum to indicate type of Error code returned by the CIRCULAR LIGHT RECOGNITION algorithm

Fields

Field	Description
TI_CLR_OBJ_RED_TRAFFIC_LIGHT	Red Traffic Light
TI_CLR_ERRORTYPE_INVALID_IMAGE_DIMS	Image dimensions are beyond supported
TI_CLR_ERRORTYPE_INVALID_ROI_DIMS	ROI dimensions are beyond image dimensions
TI_CLR_ERRORTYPE_MAXNUMRADIUS_EXCEEDED	Number of radius is beyond supported
TI_CLR_ERRORTYPE_RADIUS_BEYOND_RANGE	Radius value is beyond supported
TI_CLR_ERRORTYPE_MAXNUMCOLORS_EXCEEDED	Number of color lights exceeded the supported value
TI_CLR_ERRORTYPE_SCALING_FACTOR_BEYOND_RANGE	Hough space scaling factor is beyond supported value
TI_CLR_ERRORTYPE_TRACK_METHOD_BEYOND_RANGE	track method is beyond supported value
TI_CLR_ERRORTYPE_INVALID_LIGHT_SELCTION	Light selection method is invalid
TI_CLR_ERRORTYPE_INVALID_PRIMARY_COLOR	Primary color selction is invalid
TI_CLR_ERRORTYPE_INVALID_VOTING_METHOD	Hough space voting method is invalid
TI_CLR_ERRORTYPE_INVALID_GROUP_WIN_SIZE	Grouping window size is invalid
TI_CLR_ERRORTYPE_INVALID_MORPH_METHOD	Morphology method is invalid

5.1.10.9 TI_ CLR_ObjectType

Description

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

Fields

Field	Value	Description

Field	Value	Description
TI_CLR_OBJ_RED_TRAFFIC_LIGHT	128	Red Traffic Light
TI_CLR_OBJ_GREEN_TRAFFIC_LIGHT	129	Green Traffic Light
TI_CLR_OBJ_VEHICLE	130	Vehicle head light
TI_CLR_OBJ_MAX	131	Maximum value for CLR object type

5.1.11 Circular Light Recognition Data Structures

This section includes the following Circular Light Recognition specific extended data structures:

TI_CLR_CreateParams

TI_CLR_InArgs

TI_CLR_OutArgs

TI_CLR_objectDescriptor

TI_CLR_output

5.1.11.1 TTI_CLR_CreateParams

‖ Description

This structure defines the init-time input arguments for Circular Light Recognition 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
edma3RmLldHandle	void *	Input	Handle to EDMA3 resource manager.
maxImageWidth	int32_t	Input	Max input width of image
maxImageHeight	int32_t	Input	Max input height of image
maxNumColors	int32_t	Input	Maximum number of color lights to be detected

Field	Data Type	Input/ Output	Description
maxRadius	int32_t	Input	Maximum value for Radius
maxNumRadius	int32_t	Input	Maximum value for number of radii
minScalingFactor	int32_t	Input	Minim value for hough Space Scaling Factor, Refer TI_CLR_HoughSpaceScaling
trackingMethod	int32_t	Input	Tracking lights between frames, refer TI_CLR_TrackingMethod for list of suported tracking methods

5.1.11.2 TI_CLR_InArgs

‖ Description

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

‖ Fields

Field	Data Type	Input/ Output	Description
iVisionInArgs	IVISION_InArgs	Input	See IVISION_InArgs data structure for details.
lightSelection	int32_t	Input	list of circular lights selection method, refer TI_CLR_LightSelectionMethod
lightBrightnessThr	int32_t	Input	Bright ness level threshold for each type of light
lightColor	int32_t	Input	Primary color of each light, refer TI_CLR_PrimaryColor
lightThr1	int32_t	Input	First threshold for Color segmentation. Threshold used for Cr/V segmentation
lightThr2	int32_t	Input	Second threshold for Color segmentation. Threshold used for Cb/U segmentation
falseFilterThr	int32_t	Input	Threshold (dark pixels values) for filtering false positive
numColors	int32_t	Input	Number of colors to be searched
numRadius	int32_t	Input	Number of radii to be searched

Field	Data Type	Input/ Output	Description
`radius`	`int32_t`	Input	list of radius Values
`circleDetectionThr`	`int32_t`	Input	Detection threshold during Hough circle transform
`scalingFactor`	`int32_t`	Input	Scaling factor for Hough Space, refer TI_CLR_HoughSpaceScaling
`morphologyMethod`	`int32_t`	Input	0 : Disable Morphology, 1: 3x3 Dilation on Binary Image
`houghSpaceVotingMethod`	`int32_t`	Input	Voting method for Hough Space, Refer TI_CLR_HoughSpaceVotingMethod
`groupingWindowSize`	`int32_t`	Input	Minimum distance between two detected lights in pixels
`reserved`	`int32_t`	Input	reserved parameter for algorithm debug,shall be set zero for normal processing

5.1.11.3 TI_CLR_OutArgs

|| Description

This structure contains all the parameters which are given as an output by CLR algorithm at frame level.

|| Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_OutArgs`	Output	See `IVISION_OutArgs` data structure for details.
`numCLs`	`int32_t`	Output	Number of detected Circular lights for each color and sizes.
`numSegPixels`	`uint32_t`	Output	Number of pixels segmented for each color in the frame. This information may not be used by the application. This is just a statistics about processed frame.

5.1.11.4 TI_CLR_objectDescriptor

|| Description

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

|| Fields

Field	Data Type	Input/ Output	Description
`objType`	`uint8_t`	Output	See `TI_CLR_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
`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_CLR_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.

5.1.11.5 TI_CLR_output

‖ Description

This is the output structure given out by Circular Light Recognition module. It contains the number of objects detected and TI_CLR_MAX_DETECTIONS_PER_FRAME instances of TI_CLR_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
`errorCode`	`int32_t`	Output	Error Code returned for one process call. A value f

Field	Data Type	Input/ Output	Description
			0 indicates no error. Refer TI_CLR_errorCodes enumeration for a list of errorCodes thrown by CLR
`objDesc[]`	`TI_CLR_objectDescriptor`		See `TI_CLR_objectDescriptor` for more details

5.2 Interface Functions

This section describes the Application Programming Interfaces (APIs) used by Circular Light Recognition. 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).

5.3 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()

5.4 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()`

5.5 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 Circular Light Recognition 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, clr_tb.c available in the \test\src sub-directory.

‖ See Also

algInit(), algActivate(), process()

5.6 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 Circular Light Recognition. 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, clr_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()`

5.7 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()`