

Forward Collision Warning (FCW) using TI's TMS320C66x DSP

User Guide



May 2017

IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

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

Products

| | |
|------------------------------|--|
| 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 |
| defense | |
| 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- |
| Video & Imaging | www.ti.com/video |
| TI E2E Community | e2e.ti.com |

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright© 2017, Texas Instruments Incorporated

| | | |
|----------|--|------------|
| 1 | READ THIS FIRST | V |
| 1.1 | About This Manual | v |
| 1.2 | Intended Audience | v |
| 1.3 | How to Use This Manual | v |
| 1.4 | Related Documentation From Texas Instruments | v |
| 1.5 | Abbreviations | vi |
| 1.6 | Text Conventions | vi |
| 1.7 | Product Support | vii |
| 1.8 | Trademarks | vii |
| 2 | INTRODUCTION | 2-1 |
| 2.1 | Overview of XDAIS | 2-1 |
| 2.1.1 | XDAIS Overview | 2-1 |
| 2.2 | Overview of FCW | 2-2 |
| 2.3 | Supported Services and Features | 2-2 |
| 3 | INSTALLATION OVERVIEW | 3-3 |
| 3.1 | System Requirements | 3-3 |
| 3.1.1 | Hardware | 3-3 |
| 3.1.2 | Software | 3-3 |
| 3.2 | Installing the Component | 3-3 |
| 3.2.1 | Installing the compressed archive | 3-3 |
| 3.3 | Building Sample Test Application | 3-5 |
| 3.3.1 | Installing XDAIS tools (XDAIS) | 3-5 |
| 3.3.2 | Installing Code Generation Tools | 3-6 |
| 3.3.3 | Installing Math LIB | 3-6 |
| 3.3.4 | Building the Test Application Executable through GMAKE | 3-6 |
| 3.4 | Configuration File | 3-7 |
| 3.4.1 | Test Application Configuration File | 3-7 |
| 3.5 | Host emulation build for source package | 3-10 |
| 3.5.1 | Installing Visual Studio | 3-10 |
| 3.5.2 | Installing VLIB package for host emulation | 3-10 |
| 3.5.3 | Building source in host emulation | 3-10 |

| | | |
|------------|---|-------------|
| 3.5.4 | Running host emulation executable | 3-10 |
| 3.6 | Uninstalling the Component | 3-10 |
| 4 | SAMPLE USAGE | 4-11 |
| 4.1 | Overview of the Test Application | 4-11 |
| 4.2 | Parameter Setup | 4-11 |
| 4.3 | Algorithm Instance Creation and Initialization | 4-12 |
| 4.4 | Process Call | 4-12 |
| 4.5 | Algorithm Instance Deletion | 4-13 |
| 4.6 | Frame Buffer Management | 4-13 |
| 4.6.1 | Input and Output Frame Buffer | 4-13 |
| 5 | API REFERENCE | 5-15 |
| 5.1.1 | IVISION_Params | 5-15 |
| 5.1.2 | IVISION_Point | 5-15 |
| 5.1.3 | IVISION_Rect | 5-16 |
| 5.1.4 | IVISION_Polygon | 5-16 |
| 5.1.5 | IVISION_BufPlanes | 5-16 |
| 5.1.6 | IVISION_BufDesc | 5-17 |
| 5.1.7 | IVISION_BufDescList | 5-18 |
| 5.1.8 | IVISION_InArgs | 5-18 |
| 5.1.9 | IVISION_OutArgs | 5-18 |
| 5.1.10 | FCW Data Structures | 5-20 |
| 5.2 | Input, Output Buffer format | 5-23 |
| 5.3 | Interface Functions | 5-23 |
| 5.4 | Creation APIs | 5-24 |
| 5.5 | Initialization API | 5-26 |
| 5.6 | Control API | 5-27 |
| 5.7 | Data Processing API | 5-28 |
| 5.8 | Termination API | 5-32 |
| 6 | FREQUENTLY ASKED QUESTIONS | 6-33 |
| 6.1 | Code Build and Execution | 6-33 |
| 6.1.1 | Algorithm Related | 6-33 |

1 Read This First

1.1 About This Manual

This document describes how to install and work with Texas Instruments' (TI) FCW 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 FCW 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 FCW 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 FCW 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 |
|--------------|---|
| FCW | Forward Collision Warning |
| 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 |
| 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 (FCW Module on TMS320C66x DSP) and version number. The version number of the FCW Module is included in the Title of the Release Notes that accompanies the product release.

1.8 Trademarks

Code Composer Studio, eXpressDSP 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 FCW 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 FCW

The FCW module can be used to estimate the distance of the objects present in scene. FCW module takes input of already detected object information as one of the main input parameter. Object information needed are its image location, and width and height of object in image domain. The estimated distance given out from this module is in meters in 3D world, and it is from camera, i.e camera is assumed to be placed at origin. There are many fields in FCW_TI_output, but currently only distance is the valid output given out by this module. Ground plane information is also needed by this module, and distance is found for the objects lying on the ground. If the objects are not laying on the ground then this module cannot find its distance.

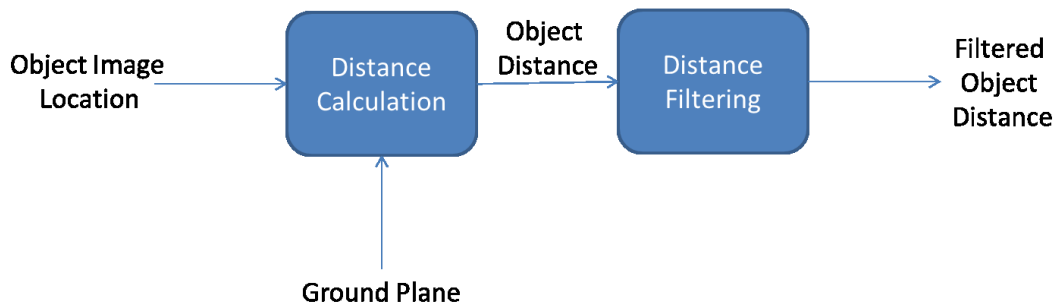


Figure 1 Block Diagram of FCW

2.3 Supported Services and Features

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

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

- ❑ Supports single precision 32 bit floating point format for out distance for objects.
- ❑ 32 maximum objects can be processed in single call of FCW module.
- ❑ Supports maximum 4000 input feature points in a single process call
- ❑ Supports user controlled parameters to control between accuracy and run time performance.

3 Installation Overview

This chapter provides a brief description on the system requirements and instructions for installing FCW 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 or TDA3x 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 FCW module:

- ❑ **Development Environment:** This project is developed using TI's Code Generation Tool 7.4.19. 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.FCW.C66x.00.01. Folder structure of this top level directory is shown in below figure.

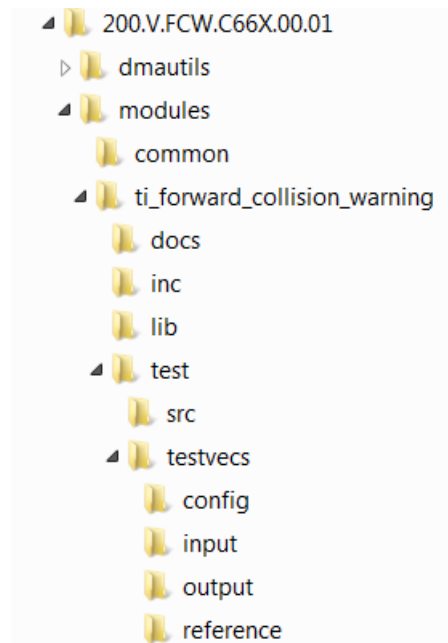


Figure 2 Component Directory Structure In case of Object Release

Table 2 Component Directories in case of Object release

| Sub-Directory | Description |
|--|---|
| \modules | Top level folder containing different DSP app modules |
| \dmautils | EDMA related APIs, not used currently. |
| \modules\common | Common files for building different DSP modules, and other common support functions needed in dsp app module. |
| \modules \ti_fcw | Top level main directory for FCW module |
| \modules \ti_forward_collision_warning\docs | User guide and Datasheet for FCW module |
| \modules \ti_forward_collision_warning\inc | Contains fcw.cfg interface file |
| \modules \ti_forward_collision_warning\lib | Contains FCW algorithm library |
| \modules \ti_forward_collision_warning\test | Contains standalone test application source files |
| \modules \ti_fcw \test\out | Contains test application .out executable |

| Sub-Directory | Description |
|--|---|
| \modules \ti_fcw \test\src | Contains test application source files |
| \modules \ti_fcw \test\testvecs | Contains config, input, output, reference test vectors |
| \modules \ti_fcw \test\testvecs\config | Contain config file to set various parameters exposed by FCW module and other parameters to control test application |
| \modules \ti_fcw \test\testvecs\input | Object information in binary file. It also contains another file .yaml containing camera intrinsic and extrinsic parameters |
| \modules \ti_fcw \test\testvecs\output | Will contain output distance information about each object in binary file. |
| \modules \ti_fcw \test\testvecs\reference | Reference binary file for each objects output information. |

3.3 Building Sample Test Application

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

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

- Refer to release notes for dependent component and their versions

3.3.1 Installing XDAIS tools (XDAIS)

XDAIS version 7.22 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 7.4.19 from the link

https://www-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 Math LIB

It can be downloaded from

<http://www.ti.com/tool/mathlib>

Set a system environment variable “MATHLIB_PATH” pointing to install directory, e.g. “C:\ti\mathlib_c66x_3_1_0_0”

3.3.4 Building the Test Application Executable through GMAKE

The sample test application that accompanies FCW 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 version 7.4.19.
- 2) Verify that you have installed XDAIS tools version 7.22.00.03
- 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_forward_collision_warning\test> gmake clean
 - b) modules\ti_forward_collision_warning\test> gmake all
- 5) The above step creates an executable file, test_object_detection_algo.out in the modules\ti_forward_collision_warning\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_forward_collision_warning\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 FCW 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 (fcw.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, fcw.cfg contains the configuration parameters required for the algorithm. The fcw.cfg file is available in the \test\testvecs\config sub-directory.

A sample fcw.cfg file is as shown.

```
#####
# Configuration Parameters For Forward Collision Warning
#####

numTestCases          = 1

#-----
#inYUVFileName = Input YUV file. Not used currently
#-----

inYUVFileName          = "../testvecs/input/img"
#-----

#outYUVFileName = Output YUV file. Not used currently
#-----

outYUVFileName          = "../testvecs/output/fcwYUVoutput"
#-----

#inObjFileName = Input object binary file
```

```

#-----
inObjFileName          = "../testvecs/input/inObjectFile.bin"
#-----
#outTTCFileName = Output object information in binary file
#-----
outTTCFileName          = "../testvecs/output/fcwOutput.bin"
#-----
#camExtPrmFileName = Camera Extrinsic information. Not used currently
#-----
camExtPrmFileName       = "../testvecs/input/out_camera_data_proj.yml"
#-----
#grndPlnPrmFileName = Input ground plane information in binary file
#-----
grndPlnPrmFileName      = "../testvecs/input/grndPlnFile.bin"
#-----
#refTTCFileName = Reference output information for each object
#-----
refTTCFileName          = "../testvecs/reference/refFcwOutput.bin"
#-----
#minObjHeight = Minimum object height in meters
#-----
minObjHeight            = 0.65
#-----
#maxObjHeight = Maximum object height in meters
#-----
maxObjHeight            = 2.60
#-----
#imgHeight = Input image height

```

```

#-----
imgHeight                = 720
#-----

#fps = input fps
#-----

fps                      = 10
#-----

#startFrame = start frame number
#-----

startFrame               = 0
#-----

#maxFrames = Total number of frames for processing
#-----

maxFrames                = 44
#-----

#maxNumObjects = Maximum number of objects in a process call
#-----

maxNumObjects            = 32
#-----

#minObjDist = Minimum distance of object in meter
#-----

minObjDist               = 0.00
#-----

#maxObjDist = Maximum distance of object in meter
#-----

maxObjDist               = 60.0

```

If you specify additional fields in the fcw.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

```
{fcw_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

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

This will build the host emulation executable under the path

```
{fcw_install_path}\test\out\ dsp_test_fcw_algo.out.exe
```

3.5.4 Running host emulation executable

Launch Microsoft Visual Studio 11.0 and open file dsp_test_fcw_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 FCW component.

4.1 Overview of the Test Application

The test application exercises the `IVISION` and extended class of the FCW 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 | ----- <code>algNumAlloc()</code> -----> ----- <code>algAlloc()</code> -----> ----- <code>algInit()</code> -----> | |
| Process Call | ----- <code>control()</code> -----> ----- <code>process()</code> -----> ----- <code>control()</code> -----> | |
| Algorithm instance deletion | ----- <code>algNumAlloc()</code> -----> ----- <code>algFree()</code> -----> | |

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, FCW 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 `fcw.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 `FCW_TI_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()`:

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

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

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. User not able to provide memory with requested in appropriate place (DARAM1 or DARAM0) shall hinder the performance.

4.4 Process Call

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

- 1) Sets the dynamic parameters (if they change during run-time) by calling the `control()` function with the `IALG_SETPARAMS` command.
- 2) Sets the input and output buffer descriptors required for the `process()` function call. The input and output buffer descriptors are obtained by calling the `control()` function with the `IALG_GETBUFINFO` command.

- 3) 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.
- 4) 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:

- 1) `algActivate()` - To activate the algorithm instance.
- 2) `control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the eight control commands.
- 3) `process()` - To call the Algorithm with appropriate input/output buffer and arguments information.
- 4) `control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the eight available control commands.
- 5) `algDeactivate()` - To deactivate the algorithm instance.

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

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

4.5 Algorithm Instance Deletion

Once `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:

- 1) `algNumAlloc()` - To query the algorithm about the number of memory records it used.
- 2) `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

Currently input buffer and output buffer is freed immediately once process call returns. Hence no locking of input and output buffers.

5 API Reference

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

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 for data memory then the pointer can be filled with NULL. If the algorithm receives 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 polygon

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 interest for the current frame to be processed in the buffer. Dimensions need to be a multiple of internal block dimensions. 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 interest for the current sub frame to be processed in the buffer. Dimensions need to be a multiple of internal block dimensions. 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::subFrameInfo is 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_Bu fPlanes | Input | Description of each plane |
| formatType | XDAS_UInt3 2 | 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 2 | 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 arguments

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 arguments

Fields

| Field | Data Type | Input/ Output | Description |
|---|-------------|------------------|---|
| Size | XDAS_UInt32 | Input | Size of the structure |
| inFreeBufIDs[IVISION_MAX_NUM_FREE_BUFFERS] | XDAS_UInt32 | Input | <p>Array of bufferId's corresponding to the input buffers that have been unlocked in the Current process call.</p> <p>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.</p> <p>If no input buffer was unlocked in the process call, inFreeBufIDs[0] will have a value of zero.</p> |
| outFreeBufIDs[IVISION_MAX_NUM_FREE_BUFFERS] | XDAS_UInt32 | Input | <p>Array of bufferId's corresponding to the Output buffers that have been unlocked in the Current process call.</p> <p>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.</p> <p>If no output buffer was unlocked in the process call, inFreeBufIDs[0] will have a value of zero.</p> |
| reserved[2] | XDAS_UInt32 | | Reserved for future usage |

5.1.10 FCW Data Structures

This section includes the following FCW specific extended data structures:

- ❑ FCW_TI_CreateParams
- ❑ TI_FCW_CamPrm
- ❑ TI_FCW_Config
- ❑ FCW_TI_InArgs
- ❑ FCW_TI_Stats
- ❑ TI_FCW_OutArgs
- ❑ FCW_TI_output
- ❑ FCW_TI_trackInfo

5.1.10.1 FCW_TI_CreateParams

|| Description

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

|| Fields

| Field | Data Type | Input/ Output | Description |
|---------------|----------------|------------------|---|
| visionParams | IVISION_Params | Input | See IVISION_Params data structure for details |
| maxNumObjects | uint16_t | Input | Maximum number of objects that can be processed by the module. |
| minObjDist | float | Input | Minimum object distance from camera to be considered by FCW algorithm. This is in meters. |
| maxObjDist | float | Input | Maximum object distance from camera to be considered by FCW algorithm. This is in meters. |
| minObjHeight | float | Input | Minimum object height of the detected object to be considered by FCW algorithm. This is in meters. |
| maxObjHeight | float | Input | Maximum object height of the detected object to be considered by FCW algorithm. This is in meters. |
| camIntPrm | float[] | Input | 3x3 floating point intrinsic camera parameters matrix is given below. [fx, 0.0, x0, 0.0, fy, y0, 0.0 |

| Field | Data Type | Input/ Output | Description |
|-------|-----------|------------------|---|
| | | | 0.0, 1.0]. Where f_x and f_y are standard scaling parameter in x and y direction. Whereas (x0, y0) is the principal point, where optic axis intersects the image plane. |

5.1.10.2 FCW_TL_InArgs

|| Description

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

|| Fields

| Field | Data Type | Input/ Output | Description |
|------------------|----------------|------------------|--|
| iVisionInArgs | IVISION_InArgs | Input | See IVISION_InArgs data structure for details. |
| camExtPrm | float[] | Input | 3x4 floating point extrinsic camera parameters matrix. Not used currently. |
| grndPlaneEq | Float[] | Input | Computed ground plane equation in the form of $A*X + B*Y + C*Z = 1$. where X,Y,Z are three dimensional co-ordinates and $A = \text{grndPlaneEq}[0]$, $B = \text{grndPlaneEq}[1]$, $C = \text{grndPlaneEq}[2]$. |
| camHeightInMeter | float | Input | Camera height from ground plane in meters. |
| camOffsetInMeter | float | Input | Camera offset from car front edge in meters. |
| fps | float | Input | Frame per second of camera capture. |
| imgHeight | int16_t | Input | Height of the input image. |

5.1.10.3 FCW_TL_Stats

|| Description

This structure reports FCW statistics, to be used only for debugging and analyzing the performance and accuracy. Currently nothing is given out as part of this structure.

|| Fields

| Field | Data Type | Input/ Output | Description |
|-------|-----------|------------------|---------------------|
| rsvd1 | uint32_t | Output | Reserved parameter. |
| rsvd1 | uint32_t | Output | Reserved parameter. |

5.1.10.4 FCW_TI_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. |
| FCWStats | FCW_TI_Stats | Output | See FCW_TI_Stats data structure for details. |

5.1.10.5 FCW_TI_output

|| Description

This is output structure given out by FCW module. It contains the number of objects detected and FCW_TI_MAX_INPUT_OBJECTS instances of FCW_TI_objectWorldInfo structure. The number of valid descriptors is governed by numObjects variable.

|| Fields

| Field | Data Type | Input/ Output | Description |
|------------------|--------------------------|------------------|---|
| numObjects | int32_t | Output | Image plane x co-ordinate |
| errorCode | int32_t | Output | Image plane Y co-ordinate |
| cameraVelocity | float | Output | X co-ordinate for 3D point, assuming current camera as origin |
| cameraLocInWorld | float | Output | Y co-ordinate for 3D point, assuming current camera as origin |
| objWorldInfo | FCW_TI_objectWorldInfo[] | Output | Z co-ordinate for 3D point, assuming current camera as origin |

5.1.10.6 FCW_TI_objectWorldInfo

|| Description

This structure defines the format of output buffer given out from FCW algorithm.

|| Fields

| Field | Data Type | Input/ Output | Description |
|-----------------|-----------|------------------|--|
| objTag | uint16_t | Output | object tag identifier to relate with input object. |
| distFrmCamera | float | Output | Distance between object and camera in meters. |
| timeToCollision | float | Output | Time to collide with the object in seconds. Not supported in current version. |
| objLocInWorld | float[] | Output | Location of object in terms of world co-ordinates.[X,Y,Z]. |
| objLocInCamera | float[] | Output | Location of object in terms of camera co-ordinates.[X,Y,Z]. |
| objVelocity | float[] | Output | Estimated velocity of object. objVelocity[0], objVelocity[1], objVelocity[2] represent velocity component in X,Y,Z direction respectively. Not supported in current version. |
| objInfo | uint32_t | Output | It can be used as the first parameter check if the object information is erroneous or not. Then only rest of the parameters may be used. Refer @FCW_TI_ObjInfo |

5.2 Input, Output Buffer format

Input and Output buffers are not image pixel buffers. Input buffer is treated as objects of the structure FCW_TI_OD_input. Output buffer is treated as one object of FCW_TI_output.

5.3 Interface Functions

This section describes the Application Programming Interfaces (APIs) used by FCW . 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 subsequent 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 FCW 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 FCW . 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 Also

See test application file, FCW_tb.c available in the \test\src sub-directory.

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 |
|----------|--------|
|----------|--------|
