Vision and Imaging Applications on EVE

User's Guide

June 18, 2018

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

Contents

1 Introduction

This document describes the details about vision and imaging applications/applet on EVE subsystem. EVE applets differ from EVE kernels as per below definition

EVE Kernels – These functions expect input and outout to be available in EVE subsystem memory. Since the EVE subsystem has limited internal memory, these functions can only be used to operate on a smaller portion of the image - mostly referred as blocks.

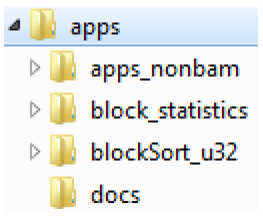
EVE Apps/Applets – These functions are built using EVE kernels and EDMA (part of EVE subsystem) to provide a frame level functionality of the kernels. These functions use EDMA as a scratch resource and relieve in scratch form. An EVE applet can be formed using single or multiple kernels. These Apps assume the entire availability of EVE subsystem across frame process boundary.. Each of the applet is provided with a test application to demonstrate its usage.

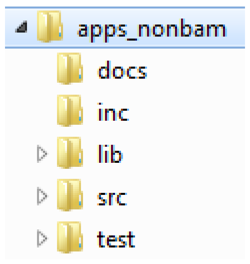
The recommended approach to create EVE applets is by using the algFramework (BAM), EVE starerware EDMA utility functions and EVE Kernels. The examples like imagePyramid_u8 and fast9_best_feature_to_front demonstrates this. The EDMA utility functions might not be able to address all the different use cases – in that case user can bypass the EDMA utility function and directly use the lower layer EDMA functions provided by starterware and create the custom EDMA utility

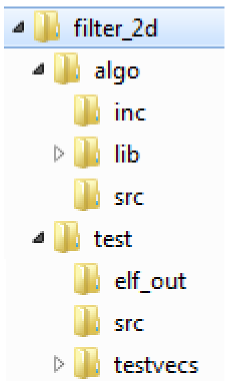
There are some apps, which are developed without algFramework (BAM) as well. These are the apps developed before evolution of BAM.

1.1 Directory Structure Overview

After installation of EVE software, below directory structure will be present on your hard disk.







Primarily below directories are relevant for the applications, which are developed without using BAM

Sub-Directory	Description
\inc	Interface header file for the applets
\lib	Library for the appltes
\src	Source code of applets
\test	Test bench code of applets
docs	Documentation on applets

The applications, which are developed using BAM, have specific directory at top level with name of the applet and rest all the relevant content is under the directory. Below is the directory structure for each app, which is developed using BAM

Sub-Directory	Description
\algo	All the files/sub-directory related to application
\test	All the files/sub-directory related to validation
algo\inc	Interface header file for the applet
algo\lib	Library of the applet
algo\src	Source code for applet (some apps might not have source because of proprietry reason)
test\testvecs	Test vectors and test configuration
test\src	Source code for test bench
test\elf_out	Executable for test bench

1.2 Apps with out using BAM

The detailed documentation of apps without using BAM is available in apps_nonbam\docs directory

1.3 Apps utilizing BAM

BAM is a recommended alg framework to develop the applets. Detailed documentation of BAM is available in algframework\docs\bam_userguide.pdf. This section describes details of each applet. It is recommended to read BAM user guide because the processing sequence of the applet is defined in that. Also there might be frequent refrences to BAM user guide in this section.

2 iVISION API Reference

This section outlines the key APIs and calling sequence of the applets. This information applies to all applets and is not repeated in each applet section. The applet sections focuses on parameters of structure and the functional behavior of applet
The interface used is called iVISION interface and it is based upon XDAIS.

Two additional APIs are defined on top of basic iALG interface exposed from XDAIS

`algControl()`

`algProcess()`

You must call these APIs in the following sequence:

`algCreate()`

`algProcess()` repeat the call as many number of slices/frame as present.

`algDelete()`

`algControl()` (if provided by the algorithm) can be called any time after calling the `algCreate()`API.

2.1 Data Processing APIs

Name

algProcess () – for processing a frame or a slice

Synopsis

```
int32_t algProcess (IVISION_Handle Handle, IVISION_InBufs *inBufs,
    IVISION_OutBufs *outBufs, IVISION_InArgs *inArgs,
    IVISION_OutArgs *outArgs));
```

Arguments

```
IVISION_Handle handle /* handle */
IVISION_InBufs * inBufs /* input Buffers */
IVISION_OutBufs * outBufs /* output Buffers */
IVISION_InArgs *inArgs /* input argumements */
IVISION_OutArgs *outArgs /* output arguments */
```

Return Value

```
int32_t /* status of the process call */
```

Description

This function does the processing of a frame or a slice/stripe of data.

Whether the processing is for a frame or a slice depends on the buffer descriptor

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 `algCreate()`.

The buffers in `inArgs` and `outArgs` are owned by the calling application.

Postconditions

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 buffers in `inArgs and outArgs` are owned by the calling application.

2.2 Control API

Name

algControl `()` – To provide/update control inofmration

Synopsis

```
int32_t algControl (IVISION_Handle Handle, IALG_Cmd cmd,
    const IALG_Params *inParams, IALG_Params *outParams);
```

Arguments

```
IVISION_Handle handle /* handle */
IALG_Cmd id /* Command id */
IALG_Params * inParams /* input Params for set kind of commands*/
IALG_Params * outParams /* output Params for get kind of commands */
```

Return Value

```
int32_t /* status of the process call */
```

Description

This function can be used to update/get parameters from the algorithm. It is an optional function and might not be implemented by all the apps. Apps will document the usage of each comamnd in detail. Application may choose to pass a pointer as NULL bassed on the the command

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 `algCreate()`.

The buffers in `params` and `status` are owned by the calling application.

Postconditions

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.

The buffers in `inParams` and `outParams` are owned by the calling application.

2.3 Data Structures

2.3.1 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

2.3.2 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)

2.3.3 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

2.3.4 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

2.3.5 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)
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

Field	Data Type	Input/ Output	Description
accessMask	XDAS_Int32	Input	Indicates how the buffer was filled by the producer, It is IVISION_ACCESSMODE_HWA or IVISION_ACCESSMODE_CPU

2.3.6 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)
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

2.3.7 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

2.3.8 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

2.3.9 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.

Field	Data Type	Input/ Output	Description
`reserved[2]`	`XDAS_UInt32`		Reserved for future usage

3 Vision Applications

This section outlines the interface and feature details of different vision applications.

3.1 Block Statistics

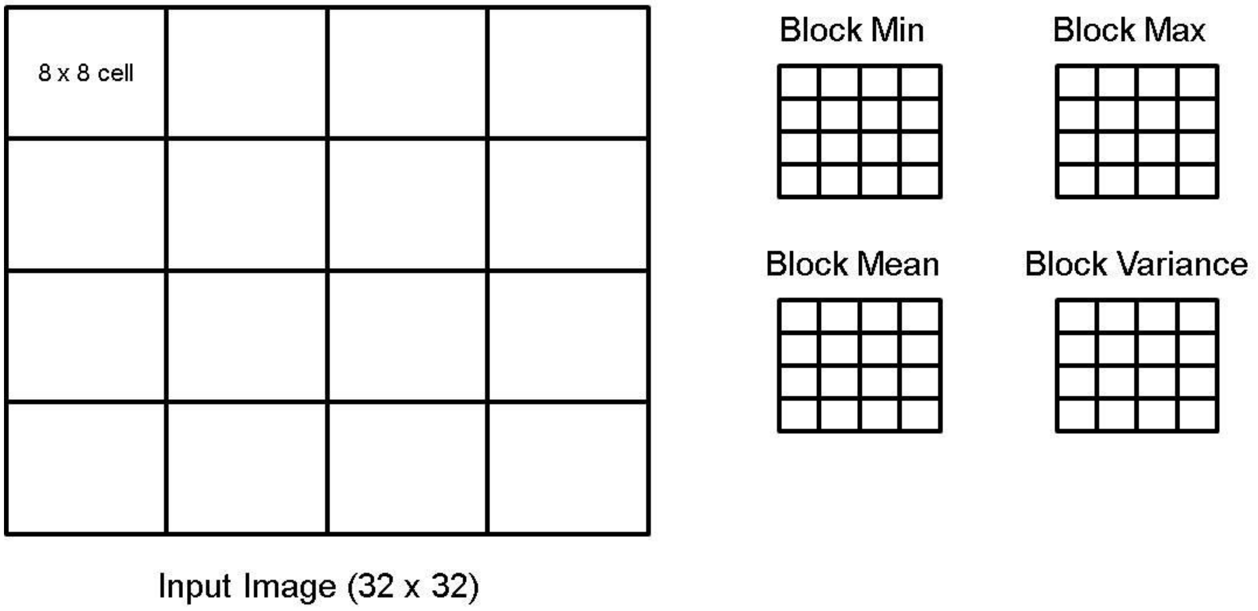
This routine accepts an 8-bit grayscale input image of size `imageWidth` by `imageHeight`. The image is divided into non-overlapping blocks of `statBlockWidth` by `statBlockHeight`. The kernel computes block statistics over these non-overlapping blocks. The following statistics are computed

1. Minima(min_B)
2. Maxima(max_B)
3. Mean(mean_B)
4. Variance(variance_A)

The applet does not perform averaging during mean and variance computations. Hence mean output reported is actually N*mean and variance is N^2*variance where N is the number of samples within a block (N = statBlockWidth*statBlockHeight). User has to divide the outputs by N and N^2 respectively to arrive at mean and variance.

For a given image frame or ROI of size M x N and block size of m x n, the ROI is divided into closely packed non-overlapping blocks/cells. There will be Mo x No such blocks where

- Mo = floor(M/m)
- No = floor(N/n)
- Example scenario for an image of size 32 x 32 with 8 x 8 cell size is shown below



3.1.1 Data Structures

3.1.1.1 BLOCK_STATISTICS_TI_CreateParams

Description

This structure defines the creation parameters for block statistics applet.

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`imageWidth`	`uint16_t`	Input	Width of the input image
`imageHeight`	`uint16_t`	Input	Height of the input image
`statBlockWidth`	`uint16_t`	Input	Block width for which the statistics has to be calculated
`statBlockHeight`	`uint16_t`	Input	Block height for which the statistics has to be calculated

3.1.1.2 Block Statistics Applet Input and Output Buffer Indices

Description

The following buffer indices are used to refer to the various input and output buffers needed by the Block Statistics applet.

Fields

Field	Data Type	Input/ Output	Description
`BLOCK_STATISTICS_TI_BUFDESC_IN_IMAGEBUFFER`	`enum`	Input	This buffer descriptor provides the input grayscale image data required by applet. The applet supports input of unsigned char type.
`BLOCK_STATISTICS_TI_BUFDESC_OUT_MIN`	`enum`	Output	This buffer descriptor points to the block minimum output buffer. The buffer is expected to be of type *uint8_t*.
`BLOCK_STATISTICS_TI_BUFDESC_OUT_MAX`	`enum`	Output	This buffer descriptor points to the block maximum output buffer. The buffer is expected to be of type *uint8_t*.
`BLOCK_STATISTICS_TI_BUFDESC_OUT_MEAN`	`enum`	Output	This buffer descriptor points to the block mean output buffer. The buffer is expected to be of type *uint16_t*.
`BLOCK_STATISTICS_TI_BUFDESC_OUT_VARIANCE`	`enum`	Output	This buffer descriptor points to the block variance output buffer. The buffer is expected to be of type *uint32_t*.

3.1.2 Constraints

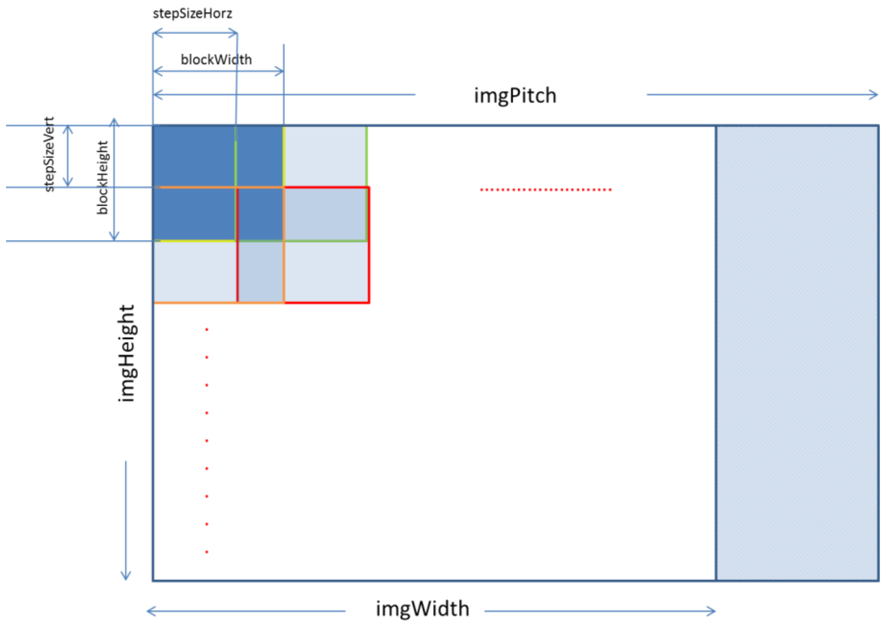
- Number of pixels in a block (blockWidth x blockHeight) <= 256
- Number of blocks vertically (blockHeight/statBlockHeight) is <= 8
- Number of blocks horizontally (blockWidth/statBlockWidth) is <= 8

- Assumptions
  - Input:
    - An 8-bit grayscale input image: (uint8_t)
  - Outputs:
    - Block Minima: (uint8_t)
    - Block Maxima: (uint8_t)
    - Block Mean: (uint16_t). Instead of outputting mean, the kernel outputs N*mean.
    - Block Variance: (uint32_t). Instead of outputting variance, the kernel provides N^2*variance.

Number of pixels within a cell is <= 256 is used to decide the precision of mean and variance

3.2 Median Filter

This routine accepts an 8-bit grayscale input image of size imageWidth by imageHeight with a stride of imagePitch. The image is divided into overlapping blocks of blockWidth by blockHeight. The block overlap can be controlled by stepSizeHorz and stepSizeVert parameters. Median output is computed over block windows, which slides by 'stepSizeHorz' pixel in horizontal and by stepSizeVert in vertical directions. Below figure shows the input picture format with all control parameters.



The median filter output will be of dimension mxn where
m = ((imageWidth - blockWidth)/stepSizeHorz) + 1 and
n = ((imageHeight - blockHeight)/stepSizeVert) + 1.

If the total number of elements in a block is even then the lower median is reported.

3.2.1 Data Structures

3.2.1.1 medianFilter_CreateParams

Description

This structure defines the creation parameters for median filter applet.

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`imageWidth`	`uint16_t`	Input	Width of the input image
`imageHeight`	`uint16_t`	Input	Height of the input image
`blockWidth`	`uint8_t`	Input	Block width for which the median has to be calculated
`blockHeight`	`uint8_t`	Input	Block height for which the median has to be calculated

Field	Data Type	Input/ Output	Description
`stepSizeHorz`	`uint16_t`	Input	Horizontal step/jump b/w blocks
`stepSizeVert`	`uint16_t`	Input	Vertical step/jump b/w blocks
`minInputBufHeight`	`uint16_t`	Output	Minimum height expected for input buffer. The algorihtm will access the extra lines after valid imageHeight during processing, but the image content there will not effect the final output.
`extraInputMem`	`uint16_t`	Output	Extra number of bytes required after the last valid input pixel (for a buffer size of imageWidth by imageHeight).
`minOutputBufStride`	`uint16_t`	Output	Minimum output buffer stride (in bytes) expected to be provided to hold the output for the given input image dimension.
`minOutputBufHeight`	`uint16_t`	Output	Minimum height of the output buffer to hold processed output for the given input image dimension.

3.2.1.2 Median Filter Applet Input and Output Buffer Indices

Description

The following buffer indices are used to refer to the various input and output buffers needed by the GLCM applet.

Fields

Field	Data Type	Input/ Output	Description
`MEDIAN_FILTER_TI_BUFDESC_IN_IMAGEBUFFER`	`enum`	Input	This buffer descriptor provides the input grayscale image data required by applet. The applet supports input of unsigned char type.
`MEDIAN_FILTER_TI_BUFDESC_OUT_IMAGEBUFFER`	`enum`	Output	This buffer descriptor points to the median filter output.

3.2.1.3 Median Filter Applet Control Command Indices

Description

This following Control Command indices are supported for Median Filter applet.
Fields

Field	Description
`TI_MEDIAN_FILTER_CONTROL_GET_BUF_SIZE`	The algorithm expects the input and output buffers to be satisfy certain contraints. These details are communicated to the user using a control call with this index. Given the input create time parameters, the control call return the buffer constraints through the output parameters of MEDIAN_FILTER_TI_CreateParams structure.

3.2.2 Constraints

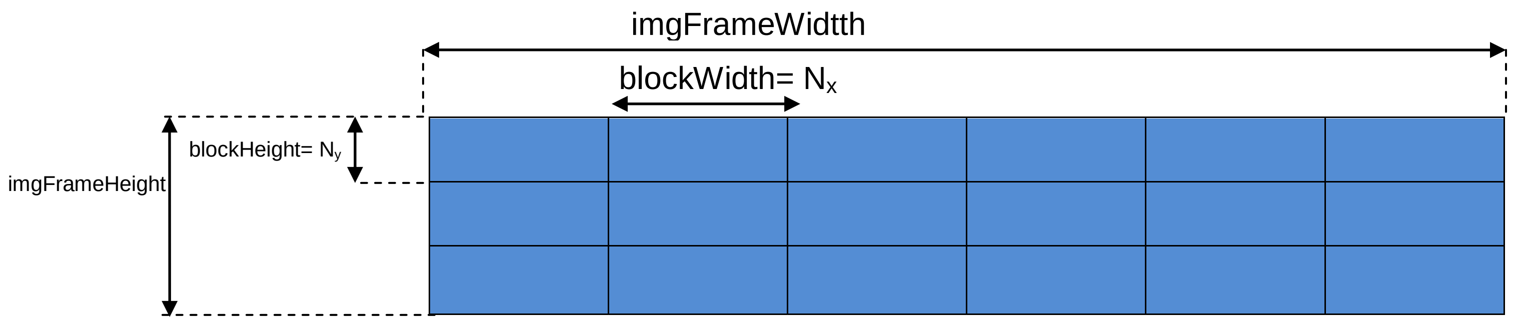
- The maximum filtering kernel size (blockHeight*blockWidth) that can be employed is restricted by the internal memory of EVE. The kernel size has to be less than ~ 16kB (16, 351 bytes to be exact).
- Apart from the special case of 3x3 dense median filtering (blockHeight = blockWidth = 3 and stepSizeHorz = stepSizeVert = 1), the current applet cannot be used for filtering with kernels of width (i.e. blockWidth) less than 9.
- The applet will be functional for all filtering kernel sizes that satisfy the above constraint. For all kernel sizes apart from 3x3 dense median filtering, the computation is based on Histogram sort approach. Hence, from a performance point of view, it is recommended to be used for large kernel median filtering.

3.3 Block sort U32

This routine sorts elements whithin multiple blocks of N 32-bits unsigned integers, with N= 64, 128, 256, 512, 1024, 2048. The blocks can be arranged in a 1-D or 2-D fashion within a frame. Indeed the structure `BLOCK_SORT_U32_TI_CreateParams` contains arguments such as `blockWidth, blockHeight, imgFrameWidth, imgFrameHeight,` which enable different layout for the data.

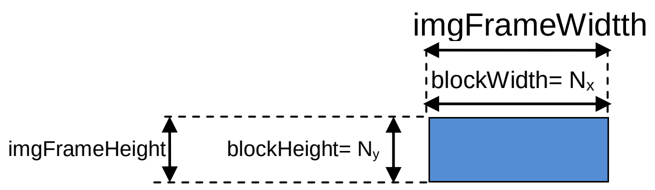
Below are some example layouts which are supported:

- 2-D frames of 2-D block with blockWidth=N_x, blockHeight= N_y with N= N_x x N_y .



`imgFrameWidth` must be a multiple of `blockWidth= Nx` and `imgFrameHeight` must be a multiple of `blockHeight= Ny`. Note that `blockHeight= Ny` can be equal to 1.

- 1 frame= 1 block with imgFrameWidth= blockWidth=N_x, imgFrameHeight= blockHeight= N_y



This layout is referred as single block processing in the remaining of the document. Single block processing is a special case of processing that is not supported with other functions. Other functions require at least 2 blocks of data to fill the pipeline. But this sorting function was customized to handle this corner case of single block processing since very often applications only need to sort a small list of elements.

Pointers of the source and destination frames residing in external memory are normally passed through the `IVISION_InBufs` and `IVISION_OutBufs` arguments of the `algProcess()` API.
However there is an exception in the case of single block processing: whenever the single block data already resides in VCOP's image buffer, it is possible to pass the location of the block to the function by setting the arguments `singleBlocksSrcAddr` and `singleBlocksDstAddr` of `BLOCK_SORT_U32_TI_CreateParams`. When `algInit() API` is called, the function recognizes that the locations are in VCOP's image buffer and does the appropriate setup such that `algProcess()` will bypass the EDMA transfers. The application must ensure that the data to be sorted is generated into the location pointed by `singleBlocksSrcAddr` and must read the results from `singleBlocksDstAddr`. By passing the EDMA transfers, whenever data is already in image buffer, allows for faster processing time. Note that single block processing also works when the single data block is in external memory but then, EDMA are used and will lengthen the processing time as it is not possible to hide data transfers behind processing when only one block is processed.

3.3.1 Data Structures

3.3.1.1 *BLOCK_SORT_U32_TI_CreateParams*

Description

This structure defines the creation parameters for block sort applet.

Fields

Field	Data Type	Input/ Output	Description
`Size`	`uint32_t`	Input	Size of the structure
`imageFrameWidth`	`uint16_t`	Input	Width of the input image
`imageFrameHeight`	`uint16_t`	Input	Height of the input image
`blockWidth`	`uint16_t`	Input	Block width of the blocks that are sorted
`blockHeight`	`uint16_t`	Input	Block height of the blocks that are sorted
`singleBlocksSrcAddr`	`uint32_t`	Input	Used for single block processing, which is automatically detected when `imageFrameWidth= blockWidth` and `imageFrameHeight= blockHeight` . Otherwise set it to 0.
`singleBlocksDstAddr`	`uint32_t`	Input	Used for single block processing, which is automatically detected when `imageFrameWidth= blockWidth` and `imageFrameHeight= blockHeight`. Otherwise set it to 0.

3.3.2 Constraints

- The product blockWidth x blockHeight must be equal to 64, 128, 256, 512, 1024 or 2048 .
- imageFrameWidth must be multiple of blockWidth.
- imageFrameHeight must be multiple of blockHeight.

3.4 Image Pyramid

This routine accepts an 8-bit grayscale input image of size srcImageWidth by srcImageHeight with a stride of srcImagePitch to produce up to 5 downscaled versions of the original image. Either 2x2 block averaging or 5x5 gaussian filtering can be used for the downscaling. The application chooses which technique will be used by setting the parameter `filterType` of the structure `IMAGE_PYRAMID_U8_TI_CreateParams`.

Each output level of the pyramid is specified through the output buffer descriptor passed to `handle->ivision->algProcess`. The following example code shows how a 3 levels pyramid for a 320x240 image is setup:

```
inBufDesc.numPlanes                          = 1;
inBufDesc.bufPlanes[0].frameROI.topLeft.x    = 0;
inBufDesc.bufPlanes[0].frameROI.topLeft.y    = 0;
inBufDesc.bufPlanes[0].width= 320;
inBufDesc.bufPlanes[0].height= 240;
inBufDesc.bufPlanes[0].frameROI.width= 320;
inBufDesc.bufPlanes[0].frameROI.height= 240;
inBufDesc.bufPlanes[0].buf = (uint8_t * )input ;

outBufDesc.numPlanes                          = 3;

outBufDesc.bufPlanes[0].frameROI.topLeft.x= 0;
outBufDesc.bufPlanes[0].frameROI.topLeft.y= 0;
outBufDesc.bufPlanes[0].width= 160;
outBufDesc.bufPlanes[0].height= 120;
outBufDesc.bufPlanes[0].frameROI.width= 160;
outBufDesc.bufPlanes[0].frameROI.height= 120;
outBufDesc.bufPlanes[0].buf = (uint16_t * )output_level1;
outBufDesc.bufPlanes[1].frameROI.topLeft.x= 0;
outBufDesc.bufPlanes[1].frameROI.topLeft.y= 0;
outBufDesc.bufPlanes[1].width= 80;
outBufDesc.bufPlanes[1].height= 60;
outBufDesc.bufPlanes[1].frameROI.width= 80;
outBufDesc.bufPlanes[1].frameROI.height= 60;
outBufDesc.bufPlanes[1].buf = (uint16_t * )output_level2;
outBufDesc.bufPlanes[2].frameROI.topLeft.x= 0;
outBufDesc.bufPlanes[2].frameROI.topLeft.y= 0;
outBufDesc.bufPlanes[2].width= 40;
outBufDesc.bufPlanes[2].height= 30;
outBufDesc.bufPlanes[2].frameROI.width= 40;
outBufDesc.bufPlanes[2].frameROI.height= 30;
outBufDesc.bufPlanes[2].buf = (uint16_t * )output_level3;

status = handle->ivision->algProcess((IVISION_Handle)handle,
                        &inBufs,&outBufs,&inArgs,(IVISION_OutArgs *)&outArgs);
```

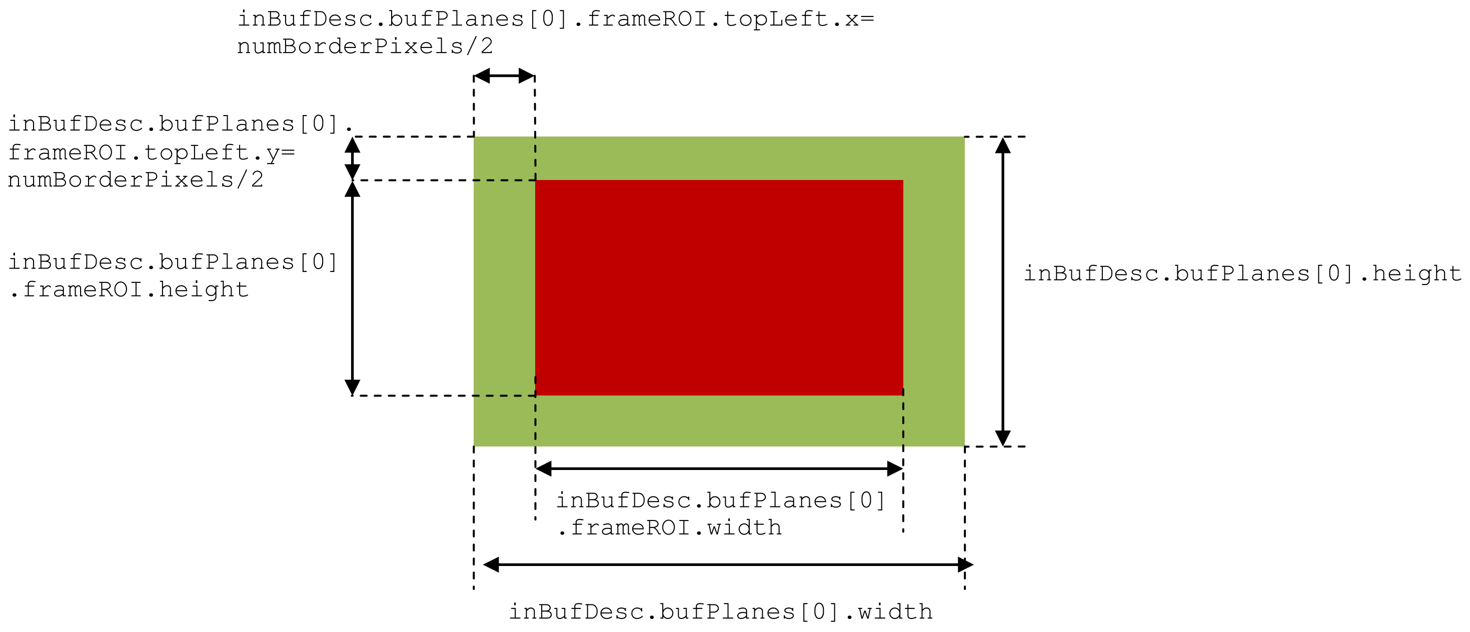
When 5x5 gaussian filter is selected, the ROI must be padded with border pixels. The application is in charge of creating these border pixels, which can be either real pixels coming from image region outside of the ROI or can be mirrored pixels from the ROI.
The number of border pixels depend on the number of pyramid levels `L` and can be calculated using this formula:

$$numBorderPixels = \sum_{l=0}^{L} 4 * 2^{l}$$

Basically, 5x5 gaussian filtering requires 4 border pixels across each dimension: 2 on the left side, 2 on the right side, 2 at the top and 2 at the bottom of the image. If there are multiple levels, gaussian filtering is applied multiple times so the number of border pixels increase along with the number of levels. However due to downscaling by a factor of 2, the number of border pixels contributed by each level increases by a factor of 2 when we use the original scale as reference. The table below shows the number of border pixels per number of pyramid levels, using the formula:

L	numBorderPixels
1	4
2	12
3	28
4	60
5	124

Basically, 5x5 gaussian filtering requires 4 border pixels across each dimension: 2 on the left side, 2 on the right side, To translate the presence of border pixels when initializing the `inBufDesc` structure, the application must take care of setting `inBufDesc.bufPlanes[0].width` and `inBufDesc.bufPlanes[0].height` such that they include these border pixels. Also, `inBufDesc.bufPlanes[0].buf` points to the upper left corner of the entire frame, border pixels included. `inBufDesc.bufPlanes[0].frameROI.topLeft.x` and `inBufDesc.bufPlanes[0].frameROI.topLeft.y` are used to indicate where the ROI actually starts and are no longer equal to 0 as it was the case in the 2x2 averaging. Please refer to the below figure for a pictorial representation of the different dimensions discussed above:



For 5x5 gaussian, the previous example code showing how a 3 levels pyramid for a 320x240 image is setup, would differ only in the way `inBufDesc` is setup, the remaining `outBufDesc` would have similar setup:

```
inBufDesc.numPlanes                          = 1;
inBufDesc.bufPlanes[0].frameROI.topLeft.x    = 14;
inBufDesc.bufPlanes[0].frameROI.topLeft.y    = 14;
inBufDesc.bufPlanes[0].width= 348;
inBufDesc.bufPlanes[0].height= 268;
inBufDesc.bufPlanes[0].frameROI.width= 320;
inBufDesc.bufPlanes[0].frameROI.height= 240;
inBufDesc.bufPlanes[0].buf = (uint8_t * )input ;
```

3.4.1 Data Structures

3.4.1.1 IMAGE_PYRAMID_U8_TI_CreateParams

Description

This structure defines the creation parameters for image pyramid applet.

Fields

Field	Data Type	Input/ Output	Description
`imgFrameWidth`	`uint16_t`	Input	Width of the input image
`imgFrameHeight`	`uint16_t`	Input	Height of the input image
`numLevels`	`uint8_t`	Input	Number of pyramid level, up to `IMGPYRAMID_MAX_NUM_LEVELS` which is 5 in current implementation
`filterType`	`uint8_t`	Input	Type of filter to be chosen. Possible options are IMAGE_PYRAMID_U8_TI_2x2_AVERAGE IMAGE_PYRAMID_U8_TI_5x5_GAUSSIAN

3.4.2 Constraints

- imgFrameWidth must be multiple of pow(2, numLevels-1)) and be greater or equal to 64.
- imgFrameHeight must be multiple of pow(2, numLevels-1) and be greater or equal to 32.
- numLevels <= 5 for 2x2 block averaging, numLevels <=4 for 5x5 gaussian

3.5 FAST9 Corner Detection

This routine accepts an 8-bit grayscale input image of size imageWidth by imageHeight with a stride of imagePitch. The output of this routine is the list of corner points detected in the image in (Y,X) packed format with Y being the first 16 bit and X being the second 16 bit in memory. Since it can accept a partial image in a bigger image, the routine also accept initial startX and startY co-ordinate position to provide the actual keypoint position in image. The pitch is provided during execution time as part buffer descriptors. Please refer ivision section for more details.
This applet can execute at multiple levels (for pyramidal usecases) at a time. Max levels supported are 4. It is important to note that the actual region processed by this applet can be lessor than the user provided input image width and height. This is done in order to get best performance from the hardware. The actual region processed will be returned as a part of outArgs. Please refer to data structures in next section for the details on input and output parameters.

3.5.1 Data Structures

3.5.1.1 FAST9_CORNER_DETECT_TI_CreateParams

Description

This structure defines the creation parameters for FAST9 Corner detection
Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`numLevels`	`uint8_t`	Input	Number of levels for which corners needs to be detected
`imgFrameWidth[]`	`uint16_t`	Input	Array of width of the input Image for all levels
`imgFrameHeight[]`	`uint16_t`	Input	Array of height of the input Image for all levels
`excludeBorderX`	`uint16_t`	input	Border to be excluded in X direction. This information will be helpful for applet to improve performance by reducing the effective imgFrameWidth. Even if this value is set to zero, the activeImgWidth can be lesser than imgFrameWidth. Refer activeImgWidth details provided below
`excludeBorderY`	`uint16_t`	Input	Border to be excluded in Y direction. This information will be helpful for applet to improve performance by reducing the effective imgFrameHeight. Even if this value is set to zero, the activeImgHeight can be lesser than imgFrameHeight. Refer activeImgHeight details provided below

3.5.1.2 FAST9_CORNER_DETECT_TI_InArgs

Description

This structure contains all the parameters which are given as an output by FAST9 corner

detect applet.
Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_InArgs`	Input	Common inArgs for all ivison based modules
`fast9Threshold[]`	`uint8_t`	Input	Threshold on difference between intensity of the central pixel and pixels of a circle around this pixel for FAST9 corner detect applet. This threshold should be provided for all levels in pyramid.
`levelMask`	`uint8_t`	Input	This indicates which of the levels in pyramid should be executed for a particular process call. Kindly refer to @IFAST9_CORNER_DECTECT_levelMask for valid values

3.5.1.3 *FAST9_CORNER_DETECT_TI_OutArgs*

Description

This structure contains all the parameters which are given as an output by FAST9 corner detect applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules
`numCorners[]`	`uint16_t`	Output	Total number of Key points (corners) detected for each input level
`activeImgWidth[]`	`uint16_t`	Output	activeImgWidth is primarily <= imgFrameWidth and decided by applet to satisfy the internal DMA and kernel requirements. This is the actual number of horizontal pixels being processed. It is exported for user as informative. This information is returned for each input level.
`activeImgHeight[]`	`uint16_t`	Output	activeImgHeight is primarily <= imgFrameHeight and decided by applet to satisfy the internal DMA and kernel requirements. This is the actual number of vertical lines being processed. It is exported for user as informative. This information is returned for each input level.

3.6 FAST9 Best Feature to Front

This routine accepts a corner list in (Y,X) packed format with Y being the first 16 bit and X being the second 16 bit in memory along with a 8-bit grayscale input image of size imageWidth by imageHeight with a stride of imagePitch. The output of this routine is the list of best N corner points (Y,X) in the same format as input. Although, input (Y,X) can be 16-bit, internally we are reducing the bit-depth of (Y,X) to 10 bit each. Hence, the (Y,X) co-ordinates has to be less than < 1024. The criterion for best N features is Fast9 score. The maximum number of features that can be processed is 2048, however user can set any value less than 2048. The number of features to be processed is accepted at

execution time but the maxFeature at create time. Internally this applet rounds up the number of features to be processed to be power of 2 (64, 128, 256,…,2048). Functionality point of view, it allows any number of features but performance will be considering as if the number of features are rounded up to next poer of 2, for example performance of 129 features will be same as 256 features. There are 2 types of suppression currently supported (4-way and 8-way suppression). In the case of 8-way suppression, max(1.3*best N, maxFeatures) features are used for suppression. It can turn out that the number of non-suppressed feature points can be less than the user desired bestN value. This information is provided as part of the outArgs.
The performance will be varying based upon number of features, the score method used, the suppression method used and details can be found in Data sheet.

3.6.1 Data Structures

3.6.1.1 FAST9_BEST_FEATURE_TO_FRONT_TI_CreateParams

Description

This structure defines the creation parameters for FAST9 Best Feature to Front

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`maxFeatures`	`uint16_t`	Input	Maximum number of features to be processed
`maxbestNFeatures`	`uint16_t`	Input	Maximum number of bestN features to be processed for 8-way suppression
`xyInIntMem`	`uint8_t`	Input	Flag to indicate if the XY list of key point location is in DDR or DMEM
`fast9Threshold`	`uint8_t`	Input	Threshold on difference between intensity of the central pixel and pixels of a circle around this pixel for FAST9 corner detection.
`scoreMethod`	`uint8_t`	Input	Method of fast9 score to be computed: FAST9_BFTF_TI_THRESH_METHOD and FAST9_BFTF_TI_SAD_METHOD are currently supported

3.6.1.2 FAST9_BEST_FEATURE_TO_FRONT_TI_InArgs

Description

This structure contains all the parameters which are given as input to FAST9 best feature to front applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_Params`	Input	Commom structure for inArgs ivision modules
`suppressionMethod`	`uint8_t`	Input	Method of suppression to be used: FAST9_BFTF_TI_SUPPRESSION_4WAY and FAST9_BFTF_TI_SUPPRESSION_8WAY are currently supported

Field	Data Type	Input/ Output	Description
`numFeaturesIn[]`	`uint16_t`	Input	Total number of Key points to be processed for each level
`bestNFeatures[]`	`uint16_t`	Input	Number of feature points locations (X,Y) to output at each level

3.6.1.3 *FAST9_BEST_FEATURE_TO_FRONT_TI_OutArgs*

Description

This structure contains all the parameters which are given as output from FAST9 best feature to front applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules
`bestNFeaturesOut[]`	`uint16_t`	Output	Number of bestN features output at each level. It can be less than the user input value bestNFeatures[ ]

3.7 Harris Corner Detection

This routine accepts an 8-bit grayscale input Region of Interest (ROI) of size imgFrameWidth by imgFrameHeight with a stride of imagePitch. The output of this routine can be of two types based on the user configuration. One is the list of corner points detected in the image in X,Y packed format with each X and Y being 16 bit quantity in memory. The order of X and Y is also configurable. The second output format is in bin packed format where each pixel is represented by a bit in outbut binary image. All set bits (having value "1") are at the location where a corner point is detected and having value "0" where no corner is detected. User also have option to configure the window size of Harris Corner detect algorithm along with the method to be used for score calculation. The details of these configurable parameters are described in later sections. The pitch is provided during execution time as part buffer descriptors. Please refer ivision section for more details.This applet also returns the actual number of horizontal and vertical pixels which are being accessed as part of outArgs. Please refer to data structures in next section for the details on input and output parameters.

The applet performs these four different processing steps:

1. Compute horizontal and vertical gradients of the image using a 3x1 filter [-1 0 1] and 1x3 filter [-1 0 1]^t respectively. This step requires a padding of 1 pixels around the frame. Please refer to VLIB's documentation of 2-D Gradient Filtering for further details on the behaviour of this step.
2. Compute 32-bits harris score from the horizontal and vertical image gradients, using a nxn averaging support window (n can take value 3,5,7). This step requires a padding of (n-1)/2 pixels around the frame. Please refer to VLIB's documentation of Harris Corner Score for further details on the behaviour of this step.
3. Compute non maximum suppression using a nxn mask around each pixel and also perform thresholding. Pixels that are not suppressed constitute the harris corners of the image. This step requires a padding of (n-1)/2 pixels around the frame. Please refer to VLIB's documentation of Non-Maxima Suppression for further details on the behaviour of this step.
4. Generate a list of (X,Y) coordinates of the detected harris corners. This is the final output produced by the applet.

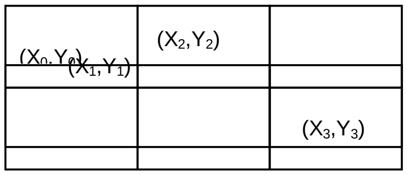
Note that if we add the padding requirement of steps 1,2,3, this amounts to a padding of 1 + (n-1)/2 +(n-1)/2 = n pixels around the frame (left, right, top, bottom). Horizontal padding is reflected by providing a pitch value at least 2n pixels greater than the ROI's image width and by setting the ROI's top left corner to start n pixels away from the top left corner of the frame.

Usage of this applet follows the calling sequence in 2 . The processing is performed by `algProcess()` for which input and output buffer descriptors must be provided. The example code below shows how these descriptors are setup to process a 320x240 ROI.

```
inArgs.subFrameInfo= 0;
inArgs.size= sizeof(IVISION_InArgs);
inBufDesc.numPlanes= 1;

/* ROI is shifted such that border pixels that extend out of the ROI are accessed during filtering */
inBufDesc.bufPlanes[0].frameROI.topLeft.x= n;
inBufDesc.bufPlanes[0].frameROI.topLeft.y= n;

/* Not that EDMA may access pixels outside of the specified image area due to some internal implementation
constraints. The area actually accessed from the topLeft corner will be given by outArgs.activeImgWidth and
outArgs.activeImgHeight. It is the application responsibiliy to ensure that inBufDesc.bufPlanes[0].width and
inBufDesc.bufPlanes[0].height are larger enough to enclose the area of dimensions activeImgWidth x
activeImgHeight whose top left corner is located at (frameROI.topLeft.x, frameROI.topLeft.y) .*/

inBufDesc.bufPlanes[0].width= 320 + 2n;
inBufDesc.bufPlanes[0].height= 240 + 2n;
inBufDesc.bufPlanes[0].frameROI.width= 320; /* same as createParams.imgFrameWidth */
inBufDesc.bufPlanes[0].frameROI.height= 240; /* same as createParams.imgFrameHeight */
inBufDesc.bufPlanes[0].buf = (uint8_t * )input;

outBufDesc.numPlanes= 1;
outBufDesc.bufPlanes[0].frameROI.topLeft.x= 0;
outBufDesc.bufPlanes[0].frameROI.topLeft.y= 0;
outBufDesc.bufPlanes[0].width= 2*createParams.maxNumCorners*sizeof(uint16_t);
outBufDesc.bufPlanes[0].height= 1;
outBufDesc.bufPlanes[0].frameROI.width= 2*createParams.maxNumCorners*sizeof(uint16_t);
outBufDesc.bufPlanes[0].frameROI.height= 1;
outBufDesc.bufPlanes[0].buf= (uint16_t * )outXY;

status = handle->ivision->algProcess((IVISION_Handle)handle,
                 &inBufs,&outBufs,&inArgs,(IVISION_OutArgs *)&outArgs);
```

The final list of (X,Y) coordinate of the detected corners is written out to the location pointed by `outBufDesc.bufPlanes[0].buf` and the number of corners detected is returned in the member `numCorners` `HARRIS_CORNER_DETECTION_32_TI_outArgs` structure passed to `algProcess()`. As the example above shows, the size of the buffer pointed by `outBufDesc.bufPlanes[0].buf` must be equal to `2*createParams.maxNumCorners*sizeof(uint16_t)` where `createParams.maxNumCorners` is the value used to initialize the member `maxNumCorners` of the applet's `HARRIS_CORNER_DETECTION_32_TI_CreateParams` structure. The reason is that the coordinates are output in (X,Y) interleaved 16-bits format.

Note that the underlying processing is performed in a block-wise fashion. The dimensions of the processing block are automatically determined by the function and are returned in the `outputBlockWidth` and `outputBlockHeight` members in the structure `HARRIS_CORNER_DETECTION_32_TI_outArgs` passed to `algProcess()`. Normally the application shouldn't care about these values except when trouble shooting performance. Typical values for `outputBlockWidth x outputBlockHeight` should be 32x30 or 36x32. Smaller output block dimensions might cause performance degradation and can be corrected by modifying the macro symbols `HARRIS_CORNER_DETECTION_BLK_WIDTH` and `HARRIS_CORNER_DETECTION_BLK_HEIGHT` in file *apps\harrisCornerDetection32\algo\src\harrisCornerDetection32_graph_int.h* to values that exactly divide `imgFrameWidth` and `imageFrameHeight`.
Also due to the block-based nature of the processing, the resulting (X,Y) coordinates list will not order the corners in a raster-scan fashion. To illustrate the difference of order between block-based processing and raster-scan processing, take the example below where 4 corners are detected. The frame is divided into processing blocks.



Normal raster scan processing will list the corners in the following order: (X_0,Y_0), (X_2,Y_2), (X_1,Y_1), (X_3,Y_3) whereas the applet's block-based processing will produce the following list: (X_0,Y_0), (X_1,Y_1), (X_2,Y_2), (X_3,Y_3)

3.7.1 Data Structures

3.7.1.1 *HARRIS_CORNER_DETECTION_32_TI_OutputFormat*

Description

The format in which Harris corner detect applet output is expected to appear.

Fields

Field	Data Type	Input/ Output	Description
`HARRIS_CORNER_DETECTION_32_TI_OUTPUT_FORMAT_LIST`	`enum`	Input	Output is the list of corners detected by this applet

Field	Data Type	Input/ Output	Description
HARRIS_CORNER_DETECTION_32_TI_OUTPUT_FORMAT_BINARY_PACK	enum	Input	Output of corners in binary packed format where a bit value of 1 indicates a corner at that pixel location and 0 indicates its not a corner Byte0 -> Bit0 (LSB of the byte in binary image) Byte7 -> Bit7 (MSB of the byte in binary image) pixels : 0 1 2 3 4 5 6 7 8 9 10 will be present as Binary : 7 6 5 4 3 2 1 0 15 14 23 12 11 10 9 8 .... and so on

3.7.1.2 HARRIS_CORNER_DETECTION_32_TI_HarrisWindowSize

Description

Supported windows size for Harris Score.

Fields

Field	Data Type	Input/ Output	Description
HARRIS_CORNER_DETECTION_32_TI_HARRIS_WINDOW_7x7	enum	Input	This enum is used to do Harris score calculation for a 7x7 window
HARRIS_CORNER_DETECTION_32_TI_HARRIS_WINDOW_5x5	enum	Input	This enum is used to do Harris score calculation for a 5x5 window
HARRIS_CORNER_DETECTION_32_TI_HARRIS_WINDOW_3x3	enum	Input	This enum is used to do Harris score calculation for a 3x3 window

3.7.1.3 HARRIS_CORNER_DETECTION_32_TI_SuppressionMethod

Description

Method to be used for Suppression of corners.

Fields

Field	Data Type	Input/ Output	Description
HARRIS_CORNER_DETECTION_32_TI_HARRIS_WINDOW_7x7	enum	Input	Use Non Maximum Suppresion in a window of size 7x7
HARRIS_CORNER_DETECTION_32_TI_HARRIS_WINDOW_5x5	enum	Input	Use Non Maximum Suppresion in a window of size 5x5
HARRIS_CORNER_DETECTION_32_TI_HARRIS_WINDOW_3x3	enum	Input	Use Non Maximum Suppresion in a window of size 3x3

3.7.1.4 HARRIS_CORNER_DETECTION_32_TI_HarrisScoreMethod

Description

Method to be used for Harris Score Calculation.

Fields

Field	Data Type	Input/ Output	Description
`HARRIS_CORNER_DETECTION_32_TI_HARRIS_SCORE_METHOD_A`	enum	Input	Score is defined as Harris Score = (Lamda1 * Lamda2) - k (Lamda1+ Lamda2)^2. This method is used to detect only corners in the image
`HARRIS_CORNER_DETECTION_32_TI_HARRIS_SCORE_METHOD_B`	enum	Input	Score is defined as Harris Score = (Lamda1+ Lamda2). This method is used to detect both corners and edges in the Image

3.7.1.5 *HARRIS_CORNER_DETECTION_32_TI_CreateParams*

Description

This structure defines the creation parameters for Harris Corner detection

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Common structure for vision modules
`imgFrameWidth`	`uint16_t`	Input	Width in bytes of the input ROI without taking into account the padding
`imgFrameHeight`	`uint16_t`	Input	Height in number of lines of the input ROI without taking into account the padding
`maxNumCorners`	`uint8_t`	Input	Maximum number of corners that the function will return in the (X,Y) coordinates list. If there are more corners detected by the NMS steps, only the first `maxNumCorners` are returned.
`harrisScoreScalingFactor`	`uint16_t`	Input	Scaling factor in Q15 format used by harris score kernel
`nmsThresh`	`int32_t`	Input	Threshold used in non-maximum suppression. For exact format of the threshold kindly refer to harris score documentation in kernels/docs/vlib_on_EVE doc
`qShift`	`uint8_t`	Input	Q shift that should be applied to detected corner points
`harrisWindowSize`	`uint8_t`	Input	Window size to be used for harris score calculation. Considers a harrisWindowSize x harrisWindowSize neighborhood to calculate Harris Score. Kindly refer to HARRIS_CORNER_DETECTION_32_TI_HarrisWindowSize for valid values

Field	Data Type	Input/ Output	Description
`harrisScoreMethod`	`uint8_t`	Input	Method to use for Harris Score calculation. Refer to HARRIS_CORNER_DETECTION_32_TI_HarrisScoreMethod for valid values
`suppressionMethod`	`uint8_t`	Input	Suppression method to be used for non maximum suppression. Kindly refer to HARRIS_CORNER_DETECTION_32_TI_SuppressionMethod for valid values
`outputFormat`	`uint8_t`	Input	The format in which output is required. Kindly refer to HARRIS_CORNER_DETECTION_32_TI_OutputFormat for supported formats

3.7.1.6 HARRIS_CORNER_DETECTION_32_TI_ControlInParams

Description

This structure defines the input control parameters passed to the `algControl()` API:

```
int32_t algControl (IVISION_Handle Handle, IALG_Cmd cmd,

    const IALG_Params *inParams, IALG_Params *outParams);
```

Used when parameter `cmd` is set to `HARRIS_CORNER_DETECTION_SET_THRESHOLD`, which instructs the `algControl() API` to use `inParams->nmsThreshold` as the current threshold value for the NMS node.

Fields

Field	Data Type	Input/ Output	Description
`algParams`	`IALG_Params`	Input	Common structure for IALG modules
`nmsThreshold`	`Int32_t`	Input	New NMS threshold value that will be used next time the `algProcess()` API is called.

3.7.1.7 HARRIS_CORNER_DETECTION_32_TI_ControlOutParams

Description

This structure defines the output control parameters passed to the `algControl()` API:

```
int32_t algControl (IVISION_Handle Handle, IALG_Cmd cmd,

    const IALG_Params *inParams, IALG_Params *outParams);
```

Used when parameter `cmd` is set to `HARRIS_CORNER_DETECTION_GET_THRESHOLD`, which instructs the `algControl() API` to return the current threshold value for the NMS node into `outParams->nmsThreshold`.

Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
`algParams`	`IALG_Params`	Input	Common structure for IALG modules
`nmsThreshold`	`int32_t`	Output	Current NMS threshold value used when the `algProcess()` API is called.

3.7.1.8 *HARRIS_CORNER_DETECTION_32_TI_outArgs*

Description

This structure contains all the parameters which are given as an output by Harris corner detect applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules
`numCorners`	`uint16_t`	Output	Total number of Key points (corners) detected
`activeImgWidth`	`uint16_t`	Output	activeImgWidth is primarily = imgFrameWidth + 14 and decided by applet to satisfy the internal DMA and kernel requirements. This is the actual number of horizontal pixels being accessed by EDMA. It is exported for user as informative
`activeImgHeight`	`uint16_t`	Output	activeImgHeight is primarily = imgFrameHeight + 14 and decided by applet to satisfy the internal DMA and kernel requirements. This is the actual number of vertical lines being accessed by EDMA. It is exported for user as informative
`outputBlockWidth`	`uint16_t`	Output	Output block width in number of pixels returned by BAM_createGraph(). That's useful information to understand performance.
`outputBlockHeight`	`uint16_t`	Output	Output block height in number of pixels returned by BAM_createGraph(). That's useful information to understand performance.

3.8 RBrief Applet

rBrief is a feature descriptor calculator. It generates a 256 bit feature vector for each feature which acts as a unique signature for the feature. This applet accepts input image for multiple levels of image pyramid, along with the list of coordinates (Y,X) of feature point and list of level id's to indicate the correspondence of feature point to the level in image pyramid. Y,X list is an list of 4 byte interger whose upper two bytes give Y coordinate and lower two byte gives X coordinate in memory. The output of this applet is list of rBrief feature descriptor of size 32 byte per feature, calculated around each of the points in the list of coordinate points.
All the execute time input to this applet comes from buffer descriptor. For further details of input and output buffers kindly refer to the irbrief.h and ivision.h interface header files. Max levels supported is 4. Please refer to data structures in next section for the details on input and output parameters.

3.8.1 Data Structures

3.8.1.1 RBRIEF_TI_CreateParams

Description

This structure defines the creation parameters for FAST9 Corner detection
Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`maxNumFeatures`	`uint8_t`	Input	Number of max Features for which the applet will be used. User can configure any value <= RBRIEF_TI_MAXNUMFEATURES
`orbPattern`	`int8_t *`	Input	This pattern defines the position of 256 pairs of src and dst to create 256 bit rBRIEF descriptor. Total size of this memory are has to be 256*4. For Exact format of this pattern - refer the User guide and test bench of this applet
`xyListInDmem`	`uint8_t`	Input	This flag indicates whether xy list is already in DMEM, if not this applet will first copy the list in DMEM. Value of 0 indidcates list is in DDR and 1 indicates the list is in DMEM.
`levelListInDmem`	`uint8_t`	input	This flag indicates whether level list is already in DMEM, if not this applet will first copy the list in DMEM. Value of 0 indidcates list is in DDR and 1 indicates the list is in DMEM

3.8.1.2 RBRIEF_TI_OutArgs

Description

This structure contains all the parameters which are given as an output by rBRIEF applet..
Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules

3.9 Pyramid LK Tracker

Pyramid LK tracker helps in tracking the corner points across the current frame, which were detected in previous frame. For instance, the two input images can be two consecutive frames of a given video sequence. This routine accepts image pyramids of two 8-bit grayscale input images namely previous frame and current frame each of which has a dimension of imageWidth x image Height with a stride of imagePitch. Maximum number of levels supported is 4 including original resolution of input image. Also, the resolution of each pyramid level is assumed half across both dimensions. That is, the resolution of other pyramid levels is as follows, (imageWidth x image Height)/4, (imageWidth x image Height)/16 & (imageWidth x image Height)/64. It also accepts two more inputs, which corresponds to corner points coordinates of two input images mentioned above. The user can has the following two options for providing an initial estimate of the corner points coordinates for the current frame, 1) Pass the same buffer for the current frame corner points coordinates, 2) Initialize the current frame corner points coordinates by adding the ego motion to the previous frame corner points coordinates. Pyramid LK tracker updates the output buffer with the tracked corner points updated coordinates for the current frame. The pitch is provided during execution time as part buffer descriptors. Pyramid LK tracker also computes the SAD based eror measure for each of the key points for the last pyramid level corresponding to original resolution. It sums up the absolute difference of the bilinear interpolated pixels of the patch windows in the previous and current frame across the original location and tracked location respectively. Please refer ivision section for more details. Please refer to the data structures in next section for the details on create time and input parameters.

3.9.1 Data Structures

3.9.1.1 PYRAMID_LK_TRACKER Macros

Description

User provides the infomration about maximum values supported by some of control parameters.

Fields

Enumeration	Description
`PYRAMID_LK_TRACKER_TI_MAXLEVELS`	Maximum number of levels supported by the applet. (Currently it is 8). Constraint:: Both width and height of lowest scale should be greater >= 9
`PYRAMID_LK_TRACKER_TI_MAXLEVELS_PER_CALL`	Maximum number of levels that can be processed in one ivion process call.
`PYRAMID_LK_TRACKER_TI_MAX_SEARCH_RANGE`	Maximum search range in each diretcion supported by Algorithm (Currently it is 18).
`PYRAMID_LK_TRACKER_TI_MAXITERATION`	Maximum Iteration supported per level (currently it is 20).

3.9.1.2 PYRAMID_LK_TRACKER_TI_CreateParams

Description

This structure defines the creation parameters for Pyramid LK tracker applet

Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`imWidth`	`uint16_t`	Input	Width of the input image in bytes
`imHeight`	`uint16_t`	Input	Height of the input image in bytes
`numLevels`	`uint16_t`	Input	Number of pyramid levels over which the corner points need to be tracked. Constraint:: Both width and height of lowest scale should be greater >= 9
`maxItersLK[PYRAMID_LK_TRACKER_TI_MAXLEVELS]`	`uint16_t`	input	Maximum number of iterations for the iterative loop of pyramid LK tracker applet. This value can be set individually for each level.
`minErrValue[PYRAMID_LK_TRACKER_TI_MAXLEVELS]`	`uint16_t`	Input	Minimum flow vector difference value at any iteration of a given pyramid level. This input is represented using Q10 format. If the motion detected for a given point is less than or equal to this threshold, then it is considered as negligible motion and thereby invokes exit from the iterative loop of pyramid LK tracker. This value can be set individually for each level.
`searchRange[PYRAMID_LK_TRACKER_TI_MAXLEVELS];`	`uint8_t`	Input	Search range in pixel for each level
`numKeyPoints`	`uint16_t`	Input	Maximum number of corners or key points that need to be tracked.

3.9.1.3 *PYRAMID_LK_TRACKER_TI_InArgs*

Description

This structure contains all the parameters which are given as an input to the Pyramid LK tracker applet during process calls.

Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_InArgs`	Input	Common inArgs for all ivison based modules
`numKeyPoints`	`uint16_t`	Input	Total number of corners or key points that need to be tracked.
`numLevels`	`uint8_t`	Input	Total number of levels to be processed in the current call.
`startLevel`	`uint8_t`	Input	Index of the first level that will be prossed in the current call. For eaxample, when user is processing total of 4 levelsthan this start value should be set to 3 and the numLevels needs to be set to 4.
`SADthreshold`	`uint16_t`	Input	Threshold for SAD to be applied to get the number of points below the this thresold

3.9.1.4 *PYRAMID_LK_TRACKER_TI_OutArgs*

Description

This structure contains all the parameters which are given as an output of the Pyramid LK tracker applet during process calls.

Fields

Field	Data Type	Input/ Output	Description
iVisionOutArgs	IVISION_OutArgs	Output	Common outArgs for all ivison based modules
numValidPoints	uint16_t	Output	Total number of valid key points after applying SAD threshold. This value is calculated only for the base level and is invalid if this applet is called without the base level.

3.9.1.5 *PYRAMID_LK_TRACKER Input Output Buffer Indices*

Description

The following buffer indices are used to refer to the various input and output buffers needed by the pyramid LK tracker algorithm.

Fields

Field	Data Type	Input/ Output	Description
PYRAMID_LK_TRACKER_TI_CURR_IMAGE_BUF_IDX	enum	Input	Buffer index for current input image data in input buffer list
PYRAMID_LK_TRACKER_TI_PREV_IMAGE_BUF_IDX	enum	Input	Buffer index for Previous input image data in input buffer list
PYRAMID_LK_TRACKER_TI_IN_KEY_POINTS_BUF_IDX	enum	Input	Buffer index for key points co-ordinates in packed XY format for previous frame in input buffer list. X is upper 16 bit and Y is lower 16 bit in memory
PYRAMID_LK_TRACKER_TI_EST_KEY_POINTS_BUF_IDX	enum	Input	Buffer index for estimated buffer co-ordinates of key points in packed XY format for current frame in input buffer list
PYRAMID_LK_TRACKER_TI_OUT_KEY_POINTS_BUF_IDX	enum	Output	Buffer index for tracked co-ordinate of key points in packed XY format for current frame in output buffer list
PYRAMID_LK_TRACKER_TI_OUT_ERR_MEASURE_BUF_IDX	enum	Output	Buffer index for error measure of key points being tracked for current frame in output buffer list

3.9.1.6 *IPYRAMID_LK_TRACKER_ErrorType Error codes*

Description

The following are error code returned by the pyramid LK tracker algorithm.

Fields

Field	Data Type	Input/ Output	Description
IPYRAMID_LK_TRACKER_ERRORTYPE_MAXLEVELS_EXCEEDED	enum	Output	The number of levels requested by user are more than supported by algorithm
IPYRAMID_LK_TRACKER_ERRORTYPE_MAXITERATION_EXCEEDED	enum	Output	The maximum number of iteration requested by user are more than supported by algorithm
IPYRAMID_LK_TRACKER_ERRORTYPE_INSUFFICIENT_BUFFERS	enum	Output	The number of input/output buffers or size of them is not sufficient for algorithm
IPYRAMID_LK_TRACKER_ERRORTYPE_NO_KEY_POINTS	enum	Output	Number of key points to be processed is requested as zero
IPYRAMID_LK_TRACKER_ERRORTYPE_NUMLEVELS_PER_CALL_EXCEEDED	enum	Output	Number of Levels to be processed is requested as zero or greater than than PYRAMID_LK_TRACKER_TI_MAXLEVELS_PER_CALL
IPYRAMID_LK_TRACKER_ERRORTYPE_START_LEVEL_EXCEEDED	enum	Output	Start Level to be processed is requested is greater than or equal to PYRAMID_LK_TRACKER_TI_MAXLEVELS
IPYRAMID_LK_TRACKER_ERRORTYPE_SEARCH_RANGE_EXCEEDED	enum	Output	Search range requiested is requested is greater than or equal to PYRAMID_LK_TRACKER_TI_MAX_SEARCH_RANGE or lesss than 1.
IPYRAMID_LK_TRACKER_ERRORTYPE_INSUFFICIENT_IM_SIZE	enum	Output	mage size (width or height) of the given level is smaller than the minimum suported image size 9

3.10 Harris Best Feature to Front

This applet prunes a list of feature points based on Harris score at each of the points. The points with maximum score are returned as output.

The routine accepts an 8-bit grayscale input image pyramid and a list of feature points provided as YX co-ordinates in packed format. Image planes and YX list for different pyramidal levels are expected to be provided in the same buffer descriptor as different buffer planes. The applet computes Harris score for each of the YX point, across all levels, using a 9x9 patch of the image around the point. It outputs a list of corner points in the image in YX packed format along with a list containing information on pyramidal level at which the feature point is present. YX packed format means that Y is upper 16 bit and X is lower 16 bit in memory

This applet can execute for a max of three pyramidal levels. Maximum number of input feature points across all levels is currently restricted to 2048.

3.10.1 Data Structures

3.10.1.1 HARRIS_BEST_FEATURE_TO_FRONT_TI_CreateParams

Description

This structure defines the creation parameters for the Harris score based best feature to front applet

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`maxNumFeaturesIn`	`uint16_t`	Input	Number of max Features for which the applet will be used. User need to make sure that total features summed up across all levels should be any value <= HARRIS_BEST_FEATURE_TO_FRONT_TI_MAXNUMFEATURES
`bestNFeaturesOut`	`uint16_t`	Input	Number of best Features user want as out User need to make sure that bestNFeaturesOut < maxNumFeaturesIn
`sensitivityParam`	`uint16_t`	Input	Value of sensitivity parameter (known as kappa in literature). The format of this is Q1.15. Which means for a value of 0.04 you should set 0.04*pow(2,15) =~ 1310.
`xyListInDmem`	`Uint8_t`	input	Flag to indicate whether the XY list input is in ARP32 Data memory or not.

3.10.1.2 IHarris_BFFT_InBufOrder

Description

User provides most of the infomration through buffer descriptor during process call.

The below enums define the purpose of the input buffer.

Fields

Enumeration	Description
HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_IN_IMAGEBUFFER	This buffer descriptor provides the actual image data required by applet. This applet works on multi level so user can provide multiple buffers. Lets say this applet is used for 3 levels of image pyramid. then user should provide inBufs->bufDesc[HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_IN_IMAGEBUFFER]->numPlanes = 3 and inBufs->bufDesc[HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_IN_IMAGEBUFFER]->bufPlanes[level] contains the details of each planes buffer pointer and dimensions
HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_IN_XY_LIST	This buffer descriptor (inBufs->bufDesc[HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_IN_XY_LIST]) should point to a buf descriptor containing XY planes for different levels. This applet works on multi level so user can provide multiple buffers. Lets say this applet is used for 3 levels of image pyramid. then user should provide inBufs->bufDesc[HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_IN_XY_LIST]->numPlanes = 3 and inBufs->bufDesc[HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_IN_XY_LIST]->bufPlanes[level] contains the pointers to that particuar level's XY list. It is user responsbility to have the X and Y list to have valid data which points in image region excluding 4 pixels boarder on each side. The applet doesn't perfrom any check for this condition and the behavior is undefined if it is not satisfied. Since it is a 1D buffer, The size of list has to be indicated by inBufs->bufDesc[HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_IN_XY_LIST]>bufPlanes[level]. width* height for a 100 points - it should be 4*100 or 400*1
HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_IN_TOTAL	This indicates total number of input buffers expected.

3.10.1.3 IHarris_BFFT_OutBufOrder

Description

User provides most of the infomration through buffer descriptor during process call.

Below enums define the purpose of out buffer.

Fields

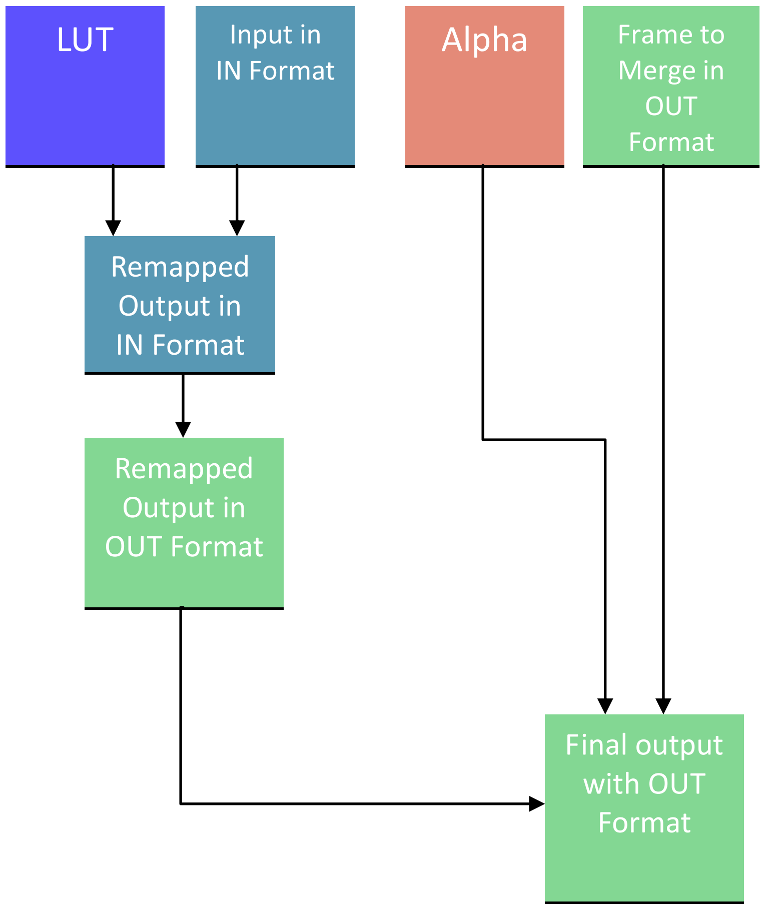
Enumeration	Description

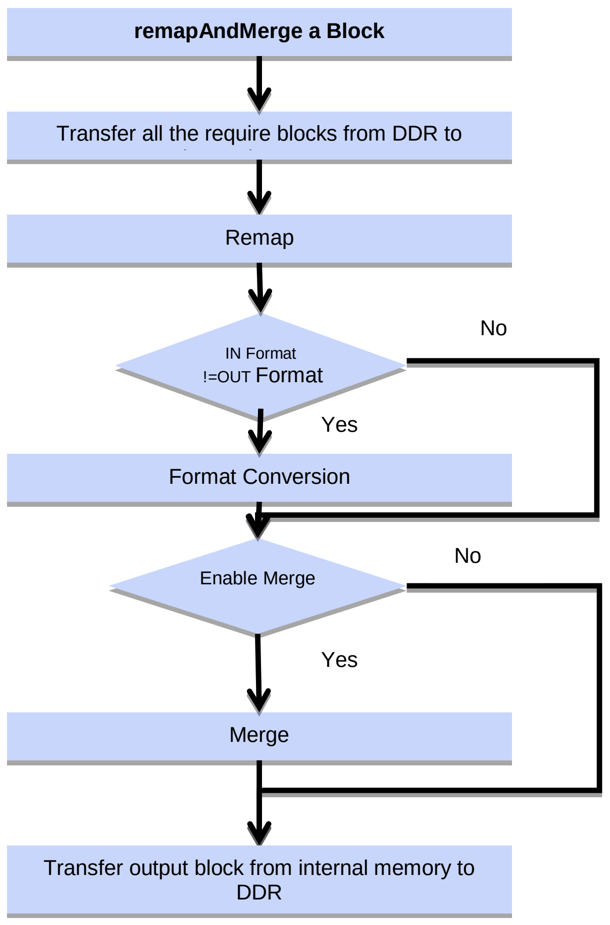
Enumeration	Description
HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_OUT_XY_LIST	This buffer descriptor (outBufs->bufDesc[HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_OUT_XY_LIST]) should point to a plane capable of holding XY list of bestNFeaturesOut. So the size of this buffer is bestNFeaturesOut*(4) bytes
HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_OUT_LEVEL_ID	This buffer descriptor (outBufs->bufDesc[HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_IN_LEVEL_ID]) should point to a plane capable of holding level corresponding to XY list. So the size of this buffer is bestNFeaturesOut*(1) bytes. There is 1:1 mapping in XY array and level array.
HARRIS_BEST_FEATURE_TO_FRONT_TI_BUFDESC_OUT_TOTAL	This indicates total number of output buffers expected.

3.11 Remap & Merge

The Remap and Merge Applet consists of two stages of processing namely Remap and Merge. During Remap, the given source image is transformed by mapping each input pixel to a new position in the destination image and thereby obtaining the remapped output, Remapped(i, j). The primary application of Remap is to correct lens distortion in the source image provided by the camera input such as Fish Eye lens, which is widely used in automotive systems. In the second stage which corresponds to merge, the remapped output is Alpha Blended with the user provided input merge frame, Merge(i, j) using the alpha frame, α(i, j). The alpha frame denotes the factor of the merge frame considered at every pixel while blending with the corresponding remapped output frame to obtain the final output frame, Out(i, j). The remapped output frame and the merge frame could be of different formats. In such cases, an additional step of format conversion is required to convert the remapped output frame format to that of merge frame provided by the user. The following flow chart depicts the overall processing flow of the remap and merge applet. For an 8 bit input image, the final output frame is obtained as follows:

$$Out(i, j) = [\alpha(i, j) * Remapped(i, j) + (16 - \alpha(i, j)) * Merge(i, j)]/16$$


LUT
Input in IN Format
Alpha
Frame to Merge in OUT Format
Remapped Output in IN Format
Remapped Output in OUT Format
Final output with OUT Format


remapAndMerge a Block
Transfer all the require blocks from DDR to
Remap
IN Format !=OUT Format
No
Yes
Format Conversion
Enable Merge
No
Yes
Merge
Transfer output block from internal memory to DDR

The function uses the user defined backmapping lookup table to find the corresponding input pixel for each output pixel. Since fractional coordinates are allowed, mapped input pixels are bilinear-interpolated from neighboring pixel values after being looked up.

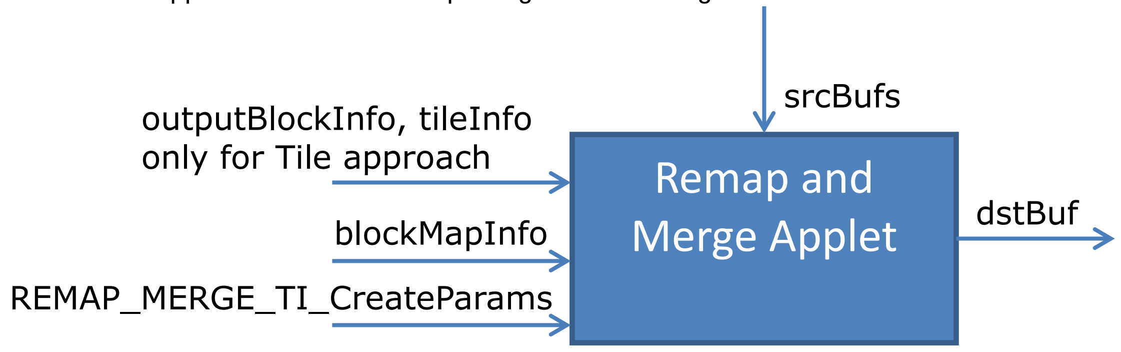
3.11.1 Current Deliverables Scope

In the current deliverables of the Remap and Merge Applet, the Remap and Alpha Blending (Merge) functionality is supported for YUV 420 SP and YUV 422 ILE with Format conversion supported between these two formats. Therefore, with a YUV 420 SP input to Remap, the user can get either a YUV 420 SP or a YUV 422 ILE Alpha Blended output depending on the Destination format the user sets. The output format should be same as that of the

Merge Frame provided to the applet. The Remap functionality alone remains supported for an extended set of input formats U8BIT, YUV422 ILE, YUV422 IBE and YUV420 SP. It is also possible to do a Remap and a Format Conversion (without Merge) for YUV 420 SP and YUV 422 ILE.

To perform Remap, two approaches are supported based on the Input block size being dynamic or constant, namely the Bounding Box approach and the Tile approach. The Applet requires the user to populate and provide the blockMapInfo, the REMAP_MERGE_TI_CreateParams and the Buffer Descriptors. In case the user enables the Tile approach for Remap, the user also needs to provide outputBlkInfo and tileInfo. The REMAP_MERGE_TI_CreateParams structure holds the configuration information for the Applet and needs to be provided by the user during Initialization. The blockMapInfo, outputBlkInfo and tileInfo can be generated using the convertMap() function. In the current deliverable, a sample utility based on convertMap() function can be used to convert a (X,Y) table describing the geometric transformation into the blockMapInfo.
The documentation for the utility is available in Appendix B: Remap Convert Table.

The Applet can be summed up using the below diagram:



3.11.2 Interface

The REMAP_MERGE_TI_CreateParams structure holds the configuration information for the Applet and needs to be provided by the user during Initialization. The structure includes the RemapParms structure which contains the parameters to configure the Remap functionality alone. The create params structure also includes two parameters, namely enableMerge and dstFormat, to determine if the alpha blending and format convert functionalities need to be enabled.
The Buffer descriptors contain the addresses of the Input Image, the Merge Frame and Alpha frame (in case Merge is enabled) and the Output Image.
Along with the REMAP_MERGE_TI_CreateParams and the Buffer descriptors, the user also needs to provide the blockMapInfo consisting of the Back Mapping Lookup Table to lookup input pixels from either the input bounding box or input tile. The pointer to the blockMapInfo is passed to the applet as REMAP_MERGE_TI_CreateParams. RemapParams.maps.blockMap pointer. The constituents of the blockMapInfo are different for both the approaches and are detailed below.

3.11.2.1 Interface for Bounding Box Approach

The blockMapInfo consists of the Back Mapping Lookup Table and the convertMapBlockInfo for every output block of output block dimensions. The Back Mapping Lookup Table should have the integral and fractional index to the Input pixel mapping to every output pixel of the output block. The convertMapBlockInfo has information of the bounding block of the Input pixels mapped to every output block. This information is mostly the position (x, y) of the Input block in the Input frame and its dimensions (width, height). The above explanation for the blockMapInfo is illustrated in the below given diagram.

convertMapBlockInfo for Output block 0
Table lookup index: 3 * output block width * output block height bytes
convertMapBlockInfo for Output block 1
Table lookup index: 3 * output block width * output block height bytes

convertMapBlockInfo for Output block 2
Table lookup index: 3 * output block width * output block height bytes
:

blockMapInfo

The Table Lookup index has the Input pixel Index for every output pixel. It is expected to be in the below format:

- 16 bits are used for storing the Integer Index. The Integral Indexes will be stacked together for every pixel of the output block size say N pixels.
- The Integral indexes will be followed by the Fractional indexes for every pixel in the output block.
- The fractional index will be an 8 bit index, of which 4 bits are allotted to x-frac and the remaining 4 bits are allotted for y-frac.

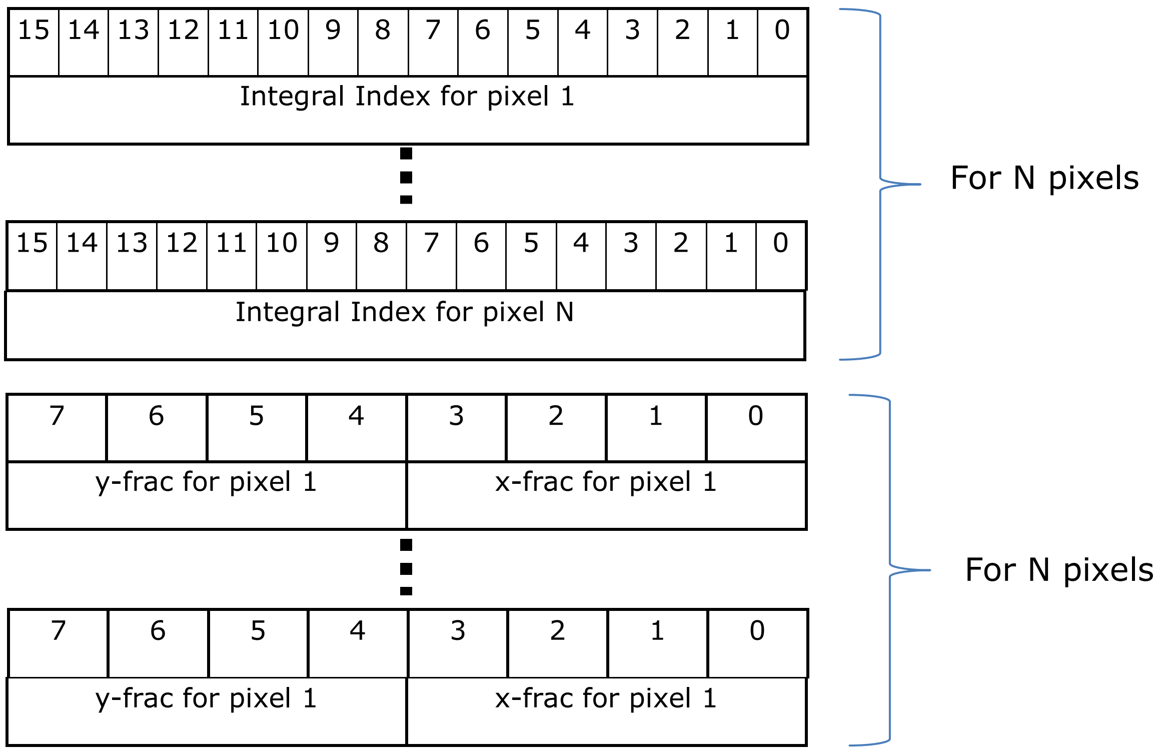


Table lookup index

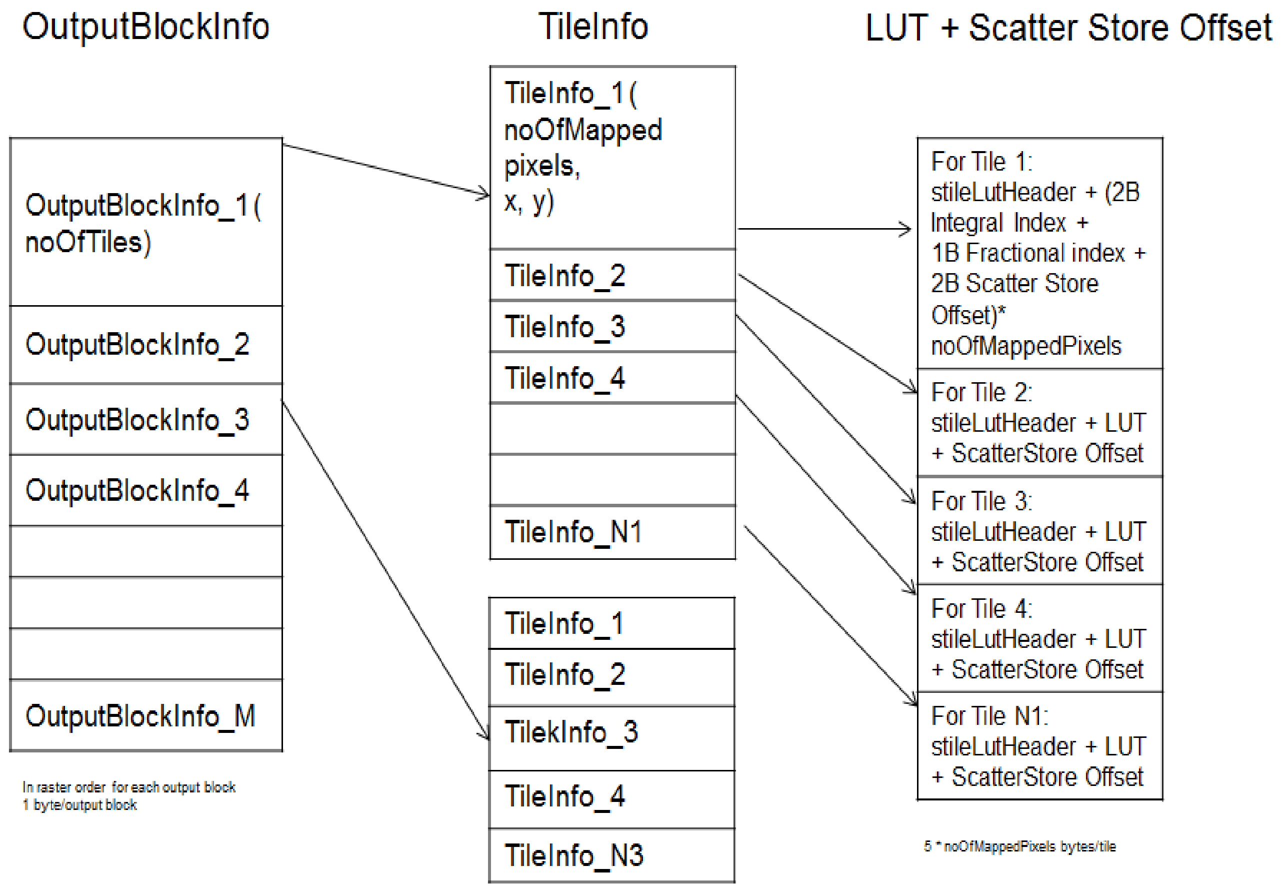
3.11.2.2 Interface for Tile Approach

For the tile approach, blockMapInfo alone falls short of information for the applet. The applet expects specific information at various stages of processing as explained below:

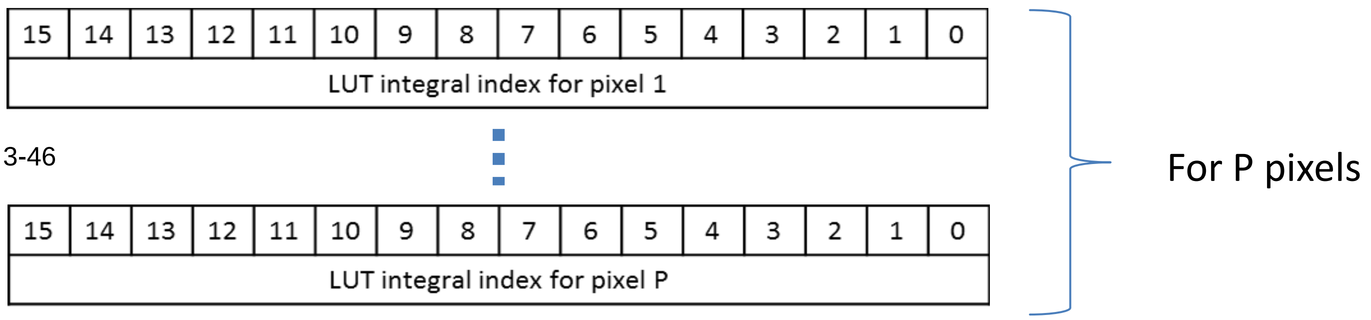
- Per Output block : outputBlockInfo
    - outputBlockInfo consists of a single field, the number of input tiles mapped to an output block, per output block.
    - If ‘m’ is the number of output blocks, then there will be ‘m’ elements of 1 byte each.
    - The pointer to the blockMapInfo is passed to the applet as REMAP_MERGE_TI_CreateParams .RemapParams.maps. outputBlkInfo.
- Per Input Tile :tileInfo

- tileInfo indicates the position (top left x, y co-ordinates) of each input tile. It also has the number of back-mapped pixels for a given input tile.
- If Ni is the number of tiles associated with the Output block 'i', and 'm' is the number of output blocks, then there will be (N1 + N2 + N3 + … + Nm) tileInfos.
- The pointer to the blockMapInfo is passed to the applet as REMAP_MERGE_TI_CreateParams .RemapParams.maps. tileInfo.

- Per Input Tile : blockMapInfo
  - blockMapInfo consists of the tileLUTHeader and the Back Mapping Lookup Table for every input tile.
  - The tileLUTHeader has information regarding the number of mapped input pixels in the tile. Of these pixels, it also denotes the number of mapped pixels which can be considered for Chroma U interpolation and those which can be considered for Chroma V interpolation.
  - The Back Mapping Lookup Table consists of 2 bytes of Integral Index + 1 byte of fractional Index + 2 bytes of Scattered store offset per back-mapped pixel. Hence, there is 5 Byte of lookup information per pixel in the Output frame as compared to the 3 Byte of information per pixel for the Bounding Box approach leading to an increase of 2 Bytes/pixel in DDR bandwidth.


OutputBlockInfo
TileInfo
LUT + Scatter Store Offset
OutputBlockInfo_1 (noOfTiles)
OutputBlockInfo_2
OutputBlockInfo_3
OutputBlockInfo_4
OutputBlockInfo_M
In raster order for each output block
1 byte/output block
TileInfo_1 (noOfMapped pixels, x, y)
TileInfo_2
TileInfo_3
TileInfo_4
TileInfo_N1
TileInfo_1
TileInfo_2
TilekInfo_3
TileInfo_4
TileInfo_N3
For Tile 1: stileLutHeader + (2B Integral Index + 1B Fractional index + 2B Scatter Store Offset)* noOfMappedPixels
For Tile 2: stileLutHeader + LUT + ScatterStore Offset
For Tile 3: stileLutHeader + LUT + ScatterStore Offset
For Tile 4: stileLutHeader + LUT + ScatterStore Offset
For Tile N1: stileLutHeader + LUT + ScatterStore Offset
5 * noOfMappedPixels bytes/tile

For a given Input tile associated to an Output block, if P is the number of pixels back-mapped from the Output block to the Input Tile. Then the Look up Table will contain the following. This information should appear in an order of even pixel positions followed by odd pixel positions. For definition of even and odd pixel position, please refer explaination of sTileLutHeader.



3.11.3 Data Structures

3.11.3.1 Size

Description

This structure defines the width and height of the buffers.

Fields

Field	Data Type	Input/ Output	Description
width	`uint32_t`	Input	Width of the block or region in pixels
height	`Size`	Input	Height of the block or region in pixels

3.11.3.2 convertMapBlockInfo

Description

This structure is a part of the BlockInfo and is associated with each block based LUT for the Input Image. The convertMap() function is expected to populate this structure.

Fields

Field	Data Type	Input/ Output	Description
inputBlockWidth	uint16_t	Input	Width of mapped Input Image Block corresponding to the LUT. For YUV 420 SP and YUV 422 formats, this field should be even
inputBlockWidthDiv2	uint16_t	Input	InputBlockWidth divided by two, to be used by the Applet
inputBlockWidthQ16	uint16_t	Input	inputBlockWidth in Q16 format
inputBlockHeight	uint16_t	Input	Height of mapped Input Image Block corresponding to the LUT. For YUV 420 SP format, this field should be even
inBlock_x	uint16_t	Input	x-coordinate of the upper left corner of the mapped Input Image block. For YUV 420 SP and YUV 422 formats, this field should be even
inBlock_y	uint16_t	Input	y-coordinate of the upper left corner of the mapped Input Image block. For YUV 420 SP and YUV 422 formats, this field should be even
Padding[10]	uint16_t	Input	Padding to make the structure size to 32 B

3.11.3.3 sTileInfo

Description

This structure is a part of the tileInfo and is associated with each input tile.. The convertMap() function is expected to populate this structure.

Fields

Field	Data Type	Input/ Output	Description
inBlock_x	`uint16_t`	Input	x-coordinate of the upper left corner of the mapped Input Image tile. This value should be even.
inBlock_y	`uint16_t`	Input	y-coordinate of the upper left corner of the mapped Input Image tile. This value should be even.
numPixels	`uint16_t`	Output	Number of backmapped Pixels from Output block to Input Tile.

3.11.3.4 *sTileLutHeader*

Description

This structure is a header to the Lookup indexes and is associated with each input tile. The convertMap() function is expected to populate this structure. Sum of numEvenPixels and numOddPixels should be equal to numPixels and it is user's responsibility to adhere this. This redundant information is requested for some specific optimization

Fields

Field	Data Type	Input/ Output	Description
numPixels	`uint16_t`	Input	number of backmapped pixels in a tile.
numEvenPixels	`uint16_t`	Input	number of backmapped even pixels in a tile. For YUV 420 SP, an even pixel refers to a pixel in an even location along width and height of output block. For YUV 422, an even pixel refers to a pixel in an even location along only the width of output block..
numOddPixels	`uint16_t`	Output	number of backmapped odd pixels in a tile. Odd pixels are the non-even pixels as defined for maxNumEvenPixelsinTile.

3.11.3.5 *sConvertMap*

Description

This structure defines the parameters for used by the convert map function.

Fields

Field	Data Type	Input/ Output	Description
srcMap	`const void *`	Input	Pointer to user defined mapping table
mapDim	`Size`	Input	Dimensions of the region of interest for which mapping table is converted to obtain blockMap and transferInfo
srcImageDim	`Size`	Input	Dimensions of the source image frame
isSrcMapFloat	`uint8_t`	Input	Indicates if the user defined mapping table is in Floating Point or Fixed Point
srcFormat	`Format`	Input	Format of the source image frame. Format is of type def uint8_t. Check the enumerated type eFormat.
outputBlockDim	`Size`	Input	Block dimensions of each output block. The region of interest is divided into output blocks of this dimension outputBlockDim
inputTileDim	`Size`	Input	Tile dimensions of each Input Tile. Valid for Tile approach.

Field	Data Type	Input/ Output	Description
outputBlkInfo	`uint8_t *`	Input	Pointer to OutputBlkInfo having the list of number of tiles for every output block. Valid for Tile approach.
tileInfo	`sTileInfo*`	Input	Pointer to tileInfo having the list tile information for every input tile mapped to the output block. Valid for Tile approach.
maxNumPixelsinTile	`uint16_t`	Input	Maximum number of backmapped pixels in a tile. Valid for Tile approach.
maxNumEvenPixelsinTile	`uint16_t`	Input	Maximum number of backmapped even pixels in a tile. For YUV 420 SP, an even pixel refers to a pixel in an even location along width and height of output block. For YUV 422, an even pixel refers to a pixel in an even location along only the width of output block. Valid for Tile approach.
maxNumOddPixelsinTile	`uint16_t`	Input	Maximum number of backmapped odd pixels in a tile. Odd pixels are the non-even pixels as defined for maxNumEvenPixelsinTile. Valid for Tile approach.
tileInfoSize	`uint32_t`	Input	Size of the tileInfo. Valid for Tile approach.
blockMap	`void *`	Output	Pointer to block partitioned map containing the blockMapInfo followed by the 16 bit TLU indices for each pixel of the output block in packed format.
qShift	`uint8_t`	Input	Number of fractional bits used to represent Q format number
maxInputBlockDim	`Size`	Input	Dimensions of the largest input block mapped to one of the output blocks. Valid for BB approach.
maxInputBlockSize	`uint32_t`	Input	Maximum size of the input block mapped to one of the output blocks to be remapped in the region of interest. Valid for BB approach. If this field is not zero, the applet enables BB approach for Remap

3.11.3.6 RemapParms

Description

This structure defines the parameters for remap merge applet.

Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
interpolationLuma	Interpolation	Input	Interpolation method for luma: Bilinear = 1 or Nearest Neighbor = 0. Check the enumerated type eInterpolation.
interpolationChroma	Interpolation	Input	Interpolation method for chroma: Bilinear = 1 or Nearest Neighbor = 0. Check the enumerated type eInterpolation.
rightShift	uint8_t	Input	Amount of right shift to convert 16 bit to 8 bit
sat_high	int32_t	Input	Upper bound saturation limit applied after shift amount of 'rightShift'
sat_high_set	int32_t	Input	Upper bound saturation value applied after shift amount of 'rightShift'
sat_low	int32_t	Input	Lower bound saturation limit applied after shift amount of 'rightShift'
sat_low_set	int32_t	Input	Lower bound saturation value applied after shift amount of 'rightShift'
maps	sConvertMap	Input	Parameters used during convert MAP function

3.11.3.7 REMAP_MERGE_TI_CreateParams

Description

This structure defines the parameters for remap merge applet.

Fields

Field	Data Type	Input/ Output	Description
visionParams	IVISION_Params	Input	Buffer descriptor list of source image frame, for luma and chroma as needed. MAX_NUM_PANES is set to 2.
remapParams	RemapParms	Input	structure defines the parameters to perform remap
enableMerge	uint8_t	Input	User can enable/disable merge functionality If enableMerge is set, the user needs to provide the merge frame and alpha frame as well
dstFormat	Format	Input	Destination format; Supported formats are as follows: U8BIT, YUV_422ILE (UYVY), YUV_422IBE (YUYV), YUV 420 SP. Check the enumerated type eFormat.

3.11.4 Constraints

- Output Block width and height must be a multiple of 2.
- Input Tile width and height must be a multiple of 2.

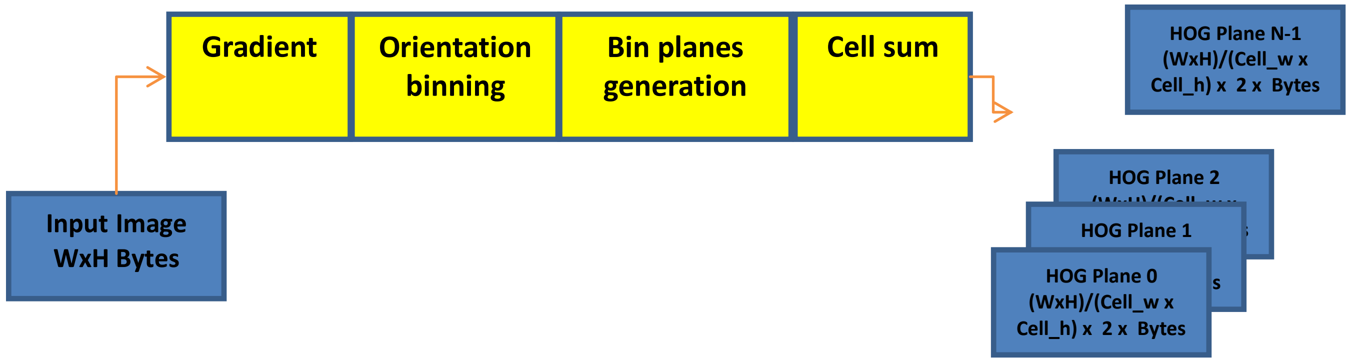
- Input Tile width should be less than 56.
- Alpha Blending is supported only for YUV 420 SP and YUV 422 ILE formats.
- Format Conversion is supported only from YUV 420 SP to YUV 422 ILE and vice-versa.

3.12 Histogram of Orientaed Gradients (HOG)

This applet accepts NV12 YUV (420 semi planner) and outputs HOG for given cell size. The below block diagram gives high-level functionalities that this applet is performing. This Applet can generate HOG plane for multiple scales together, In this case the user has to provide re-sized NV12 YUV for each scale.

To re-use the computation, this applet first performs quantum sum of the bin planes and optionaly forms cell sum if the quantum size is not equal to cell size. The quantum sum size calculated using below formula

Quantum_size = `HCF(cellSize, sreachStep, (blockSize - blockOverlap))`



The implementation is more driven from pedestrian detection use case so some of the structures and variable name indicate pedestrian detection, but this applet can be used for HOG feature computation for any other object detection as well.

3.12.1 Data Structures

3.12.1.1 PD_FEATURE_PLANE_COMPUTATION_CreateParams

Description

This structure defines the creation parameters for HOG applet

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
imgFrameWidth	`uint16_t`	Input	Max input width of image
imgFrameHeight	`uint16_t`	Input	Max input height of image
leftPadPels	`uint16_t`	Input	Number of pixels padded in the left of the original image
topPadPels	`uint16_t`	Input	Number of pixels padded in the top of the original image
cellSize	`uint8_t`	input	cell Size in Pixels for HOG computation
blockSize	`uint8_t`	Input	Block size in Pixels.
blockOverlap	`uint8_t`	Input	Block overlap in Pixels.

Field	Data Type	Input/ Output	Description
sreachStep	`uint8_t`	Input	Object model sreach step size (sliding window offset) in pixels.
scaleSteps	`uint8_t`	Input	Number of scales per octave.
maxNumScales	`uint8_t`	Input	Maximum number of scales to be processed.
numBins	`uint8_t`	Input	Number of Gradient orientation bins.
gradMagMethod	`uint8_t`	Input	Gradient Magnitude comaputation Method, Supported Values are TI_PD_GRADIENT_MAGNITUDE_SAT_8BI TS TI_PD_GRADIENT_MAGNITUDE_9BITS.
enableCellSum	`uint8_t`	Input	Parameter to enable cell sum if the quantum sum generated is not equal to the cell size.
scaleRatioQ12	`uint16_t`	Input	scale ratio that needs to be used between sucessive levels in Q12 format
additionalPlaneFLag	`uint32_t`	Input	Deatils about addition planes other than HOG planes bit - 0 Manitude plane | bit - 1 Y Plane | bit - 2 U and V Plane | bit 3-31 Unused|
outPutBufSize	`uint32_t`	Output	This is output from control call. Gives the number bytes in the output buffer.
outFormat	`uint8_t`	Input	0 - Planar Output, 1 - De-interleaved Output
scaleParams	`scalePrams_t`	Input	Information about each input scale

3.12.1.2 PD_FEATURE_PLANE_COMPUTATION_outputMetaData

Description

This structure contains all the meta data containing informtion about the feature output (num scales, size, data layout etc)

Fields

Field	Data Type	Input/ Output	Description
size	`uint32_t`	Output	Size of this stucture to validate the version check
outBufSize	`uint32_t`		Total output buffer size in bytes
numScales	`uint8_t`	Output	Total number of (pyramid levles) fearture scales in this output.

Field	Data Type	Input/ Output	Description
numPlanes	`uint8_t`	Output	Number of scales in each scale.
outFormat	`uint8_t`	Output	output layout information.
leftPadPels	`uint16_t`	Output	Number of pixels padded in the left of the original image
topPadPels	`uint16_t`	Output	Number of pixels padded in the top of the original image
scaleInfo	`PD_FEATURE_PLANE_COMPUTATION_scaleMetaData`	Output	Information about each scale feture plane.

3.12.1.3 *PD_FEATURE_PLANE_COMPUTATION_scaleMetaData*

Description

This structure contains all the meta data containing informtion about the eache scale feature (width height etc)

Fields

Field	Data Type	Input/ Output	Description
scaleOffset	`uint32_t`	Output	Offset from the base output pointer
orgImCols	`uint16_t`	Output	Image width in pixels.
orgImRows	`uint16_t`	Output	Image height in pixels
imCols	`uint16_t`	Output	Active image width in pixels.
imRows	`uint16_t`	Output	Active image height in pixels
startX	`uint16_t`	Output	Horizontal offset for active image.
startY	`uint16_t`	Output	Vertical offset for active image.
featCols	`uint16_t`	Output	Number of valid features in each row of a feature plane.
featRows	`uint16_t`	Output	Number of feature rows per feture plane.
featPitch	`uint16_t`	Output	Offset between each line of a feture plane (in Bytes).

Field	Data Type	Input/ Output	Description
planeOffset	`uint32_t`	Output	Offset between each feture plane (in Bytes).

3.12.1.4 scalePrams_t

Description

This structure contains the information about each scale

Fields

Field	Data Type	Input/ Output	Description
orgWidth	`uint16_t`	Input	Original Image width
orgHeight	`uint16_t`	Input	Original Image height
width	`uint16_t`	Input	Active/processing image width.
height	`uint16_t`	Input	Active/processing image height.
x	`uint16_t`	Input	Horizontal offset for the active image
y	`uint16_t`	Input	Vertical offset for the active image

3.12.1.5 PD_FEATURE_PLANE_COMPUTATION_InArgs

Description

This structure contains the parameters which controls the algorithm at process call level

Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_InArgs`	Input	Commom structure for vision modules
numScales	`uint8_t`	Input	Number of scales to be processed in this process call

3.12.1.6 HOG Applet Input Output Buffer Indices

Description

The following buffer indices are used to refer to the various input and output buffers needed by the HOG applet.

Fields

Field	Data Type	Input/ Output	Description
PD_FEATURE_PLANE_COMPUTATION_BUFDESC_IN_IMAGEBUFFER	enum	Input	Buffer index for current input image data in input buffer list
PD_FEATURE_PLANE_COMPUTATION_BUFDESC_OUT_FEATURE_PLANES_BUFFER	enum	Output	Buffer index for Previous input image data in input buffer list

3.12.1.7 HOG Gradient Magnitude Methods

Description

The following enum defines the gradient computations supported.

Fields

Field	Data Type	Input/ Output	Description
TI_PD_GRADIENT_MAGNITUDE_SAT_8BITS	Enum	Input	abs(x) + abs(y) staurated to 8 bit
TI_PD_GRADIENT_MAGNITUDE_9BITS	Enum	Output	abs(x) + abs(y) 9 bit

3.13 Gray-Level Co-occurrence Matrix (GLCM)

This applet computes gray-level co-occurrence matrices or gray tone spatial dependency matrices for a given 8-bit input image for texture analysis. The applet first performs an image binning before voting into the GLCM matrix. The parameter 'numLevels' specifies the number of gray-levels to use when scaling the grayscale values in input. For example, if numLevels is 8, applet scales the values in input from value 0 to value 7. The number of gray-levels ('numLevels') also determines the size of the output gray-level co-occurrence matrix. Any pixel value less than parameter 'lowPixelValue' is clipped to '0', whereas any pixel value higher than 'hiPixelVal' is clipped to 'numLevels-1'. All values in between are uniformly quantized.

The applet allows analyzing for each direction of analysis by specifying a pair of offsets (rowOffset, colOffset) between the current pixel and neighbouring pixel.

The applet allows analyzing multiple directions of analysis by passing an array of rowOffsets and colOffsets. The parameter 'numDirections' allows user to calculate co-occurrence matrix for multiple angle in single API call. Note that clubbing multiple directions of ananlysis, leads to an approximate GLCM matrix as it would lead to ignoring of few input image rows at the bottom and few image columns at the right. This mode is optimized for reduced data bandwidth.

3.13.1 Data Structures

3.13.1.1 GLCM_TI_CreateParams

Description

This structure defines the creation parameters for the GLCM applet

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`imageWidth`	`uint16_t`	Input	Width in bytes for the input image
`imageHeight`	`uint16_t`	Input	Height in number of lines for the input image
`loPixelVal`	`uint8_t`	Input	Lowest pixel value in the image. All pixels less than loPixelVal will be put into the first bin (0).
`hiPixelVal`	`uint8_t`	input	Highest pixel value in the image. All pixels more than hiPixelVal will be binned into the last bin (numLevels-1).
`numLevels`	`uint16_t`	Input	Number of gray-levels to be used for GLCM computation, The maximum numLevels permitted by the applet is GLCM_TI_MAXNUMLEVELS

Field	Data Type	Input/ Output	Description
`numDirections`	`uint16_t`	Input	Number of directions over which analysis need to be performed. At a time a maximum of GLCM_MAX_NUM_DIRECTIONS directions can be analysed together.The directions of analysis is specified as a pair of (row offset, column offset). Clubbing multiple directions of analysis together can lead to few pixels in the right and bottom border not voting into the output GLCM.
`rowOffset`	`uint8_t`	Input	Array of number of rows between the pixel of interest and its neighbor. The array should contain as many elements as numDirections.
`colOffset`	`uint8_t`	input	Array of number of columns between the pixel of interest and its neighbor. The array should contain as many elements as numDirections.

3.13.1.2 GLCM Applet Input and Output Buffer Indices

Description

The following buffer indices are used to refer to the various input and output buffers needed by the GLCM applet.

Fields

Field	Data Type	Input/ Output	Description
`GLCM_TI_BUFDESC_IN_IMAGEBUFFER`	`enum`	Input	This buffer descriptor provides the input grayscale image data required by applet. The grayscale image is binned into `numLevels` gray-levels before voting into the gray-level co-occurrence matrix
`GLCM_TI_BUFDESC_OUT_GLCM`	`enum`	Output	This buffer descriptor points to the output GLCM matrix. The GLCM matrixes for multiple angles of analysis are arranged consecutively in memory. Each GLCM matrix is of size `numLevels x numLevels.` Each entry of the matrix is 16-bit precision

3.13.2 Constraints

Output data will be clipped to 16 bit.

Range of rowOffset, and colOffset shall be [-16, 16].

Output matrix is not be symmetrical i.e. GLCM(i, j) and GLCM(j, i) are different.

Only four angles (0, 45, 90, 135) are validated

Minimum input image height and width shall be 16

numLevels should be less than FLOOR(110/numDirections)

3.14 Filter 2D

This applet implements a generic filter. The routine accepts a YUV 420 input image and along with the filter mask that needs to be applied. The output dimension of this applet is lesser than input dimension by the filter width and height. It also supports enabling disabling contrast stretching.

3.14.1 Data Structures

3.14.1.1 FILTER_2D_CreateParams

Description

This structure defines the creation parameters for the filter applet

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`filterCoefWidth`	`uint8_t`	Input	Width of the filter coefficient mask
`filterCoefHeight`	`uint8_t`	Input	Height of the filter coefficient mask
`filterCoef`	`uint8_t*`	Input	Pointer Pointing to filter coefficients array of filterCoefWidth x filterCoefHeight. For separable filter first filterCoefWidth elements will corrresponds to filter in Horizontal direction and last filterCoefHeight elements will corrresponds to filter in Vertical direction
`separableFilter`	`uint8_t`	input	A value of 1 indicates filter is separable and 0 indicates its non-separable. Currently supported method is seeprable Filter only
`enableContrastStretching`	`uint8_t`	input	A value of 1 will enable contrast stretching for Y component of image.
`minVal`	`uint8_t`	input	This parameter is only used when enableContrastStretching = 1. This is the minimum pixel value to be used for stretching the contrast
`maxVal`	`uint8_t`	input	This parameter is only used when enableContrastStretching = 1. This is the maximum pixel value to be used for stretching the contrast
`minPercentileThreshold`	`uint8_t`	input	This parameter is only used when enableContrastStretching = 1. This is the percentage to pixels used to determine the minimum value from histogram. For example a value of 1 means while calculating the minimum value of histogram we will find the minimum value which is greater than 1% of pixels

Field	Data Type	Input/ Output	Description
maxPercentileThreshold	uint8_t	input	This parameter is only used when enableContrastStretching = 1. This is the percentage to pixels used to determine the maximum value from histogram. For example a value of 99 means while calculating the maximum value of histogram we will find the maximum value which is lessor than 99% of pixels

3.14.1.2 *IFILTER_2D_InBufOrder*

Description

User provides most of the infomration through buffer descriptor during process call.

The below enums define the purpose of the input buffer.

Fields

Enumeration	Description
FILTER_2D_TI_BUFDESC_IN_IMAGEBUFFER	This buffer descriptor provides the actual image data required by algorithm. This buffer is expected to be in NV12 format with plane 0 as Luma and plane 1 as Chroma component.
FILTER_2D_TI_BUFDESC_IN_TOTAL	This indicates total number of input buffers expected.

3.14.1.3 *IFILTER_2D_OutBufOrder*

Description

User provides most of the infomration through buffer descriptor during process call.

Below enums define the purpose of out buffer.

Fields

Enumeration	Description
FILTER_2D_TI_BUFDESC_OUT_IMAGE_BUFFER	This buffer will return filtered image in NV12 format with plane 0 as Luma and plane 1 as Chroma component.
FILTER_2D_TI_BUFDESC_OUT_TOTAL	This indicates total number of output buffers expected.

3.14.1.4 *FILTER_2D_InArgs*

Description

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

Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
iVisionInArgs	IVISION_InArgs	Input	Commom structure for in args of ivision modules
minVal	uint8_t	input	Minimum value of histogram above a certain percen to be applied for current frame . This field is only valid if enableContrastStretching is set to 1 during create time
maxVal	uint8_t	input	Maximum value of histogram below a certain percent to be applied for current frame. This field is only valid if enableContrastStretching is set to 1 during create time

3.14.1.5 *FILTER_2D_OutArgs*

Description

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

Fields

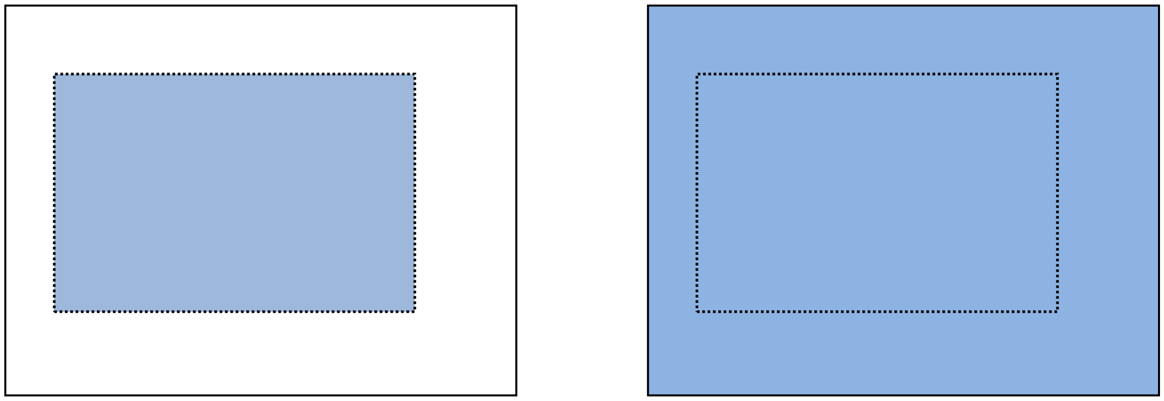
Field	Data Type	Input/ Output	Description
iVisionOutArgs	IVISION_OutArgs	output	Commom structure for in args of ivision modules
minVal	uint8_t	output	Minimum value of histogram above a certain percent for the current frame processed. This field is only valid if enableContrastStretching is set to 1 during create time
maxVal	uint8_t	output	Maximum value of histogram below a certain percent for the current frame processed. This field is only valid if enableContrastStretching is set to 1 during create time

3.14.2 Constraints

Current implementation is validated for only seperable filter. For contrast stretching it is required that image width is a multiple of 16 (2 times SIMD width)

3.15 YUV Padding

This applet accepts NV12 YUV (420 semi planner) and pads (extends) pixels in all four directions for user requested amount of pixels. It uses VCOP and EDMA for left and right padding. It uses EDMA and DMEM for top and bottom padding.



3.15.1 Data Structures

3.15.1.1 YUV PADDING Input and output Buffer IDs

Description

These macros define the IDs for input and output buffers with their properties

Fields

Field	Description
`YUV_PADDING_TI_IN_IMAGE_BUF_IDX`	Buffer index for input image data. This buffer width has to be greater than or equal to the processing width aligned to 64. This buffer height has to be greater than or equal to the processing height aligned to 64
`YUV_PADDING_TI_OUT_IMAGE_BUF_IDX`	Buffer index for output image data. Both input and output shall point to same memory region with an offset for in place padding use case. This buffer width has to be greater than or equal to the processing width aligned to 64 + left and right padding. This buffer height has to be greater than or equal to the processing height aligned to 64 + top and bottom padding

3.15.1.2 YUV_PADDING_TI_CreateParams

Description

This structure defines the creation parameters for YUV PAddign applet

Fields

Field	Data Type	Input/ Output	Description

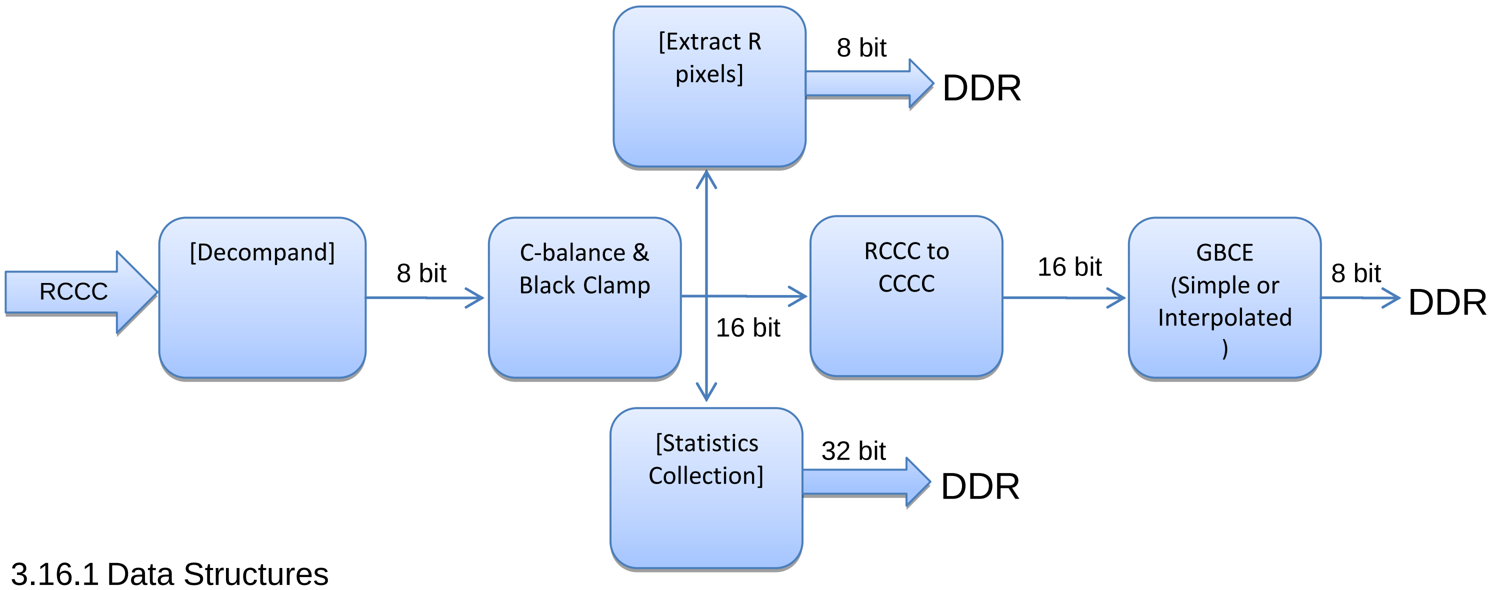
Field	Data Type	Input/ Output	Description
visionParams	IVISION_Params	Input	Commom structure for vision modules
maxImageWidth	uint16_t	Input	Maximum image width supported
topPadding	uint16_t	Input	Number of row to be padded on top
leftPadding	uint16_t	Input	Number of columns to be padded on left
rightPadding	uint16_t	input	Number of columns to be padded on right
BottomPadding	uint16_t	Input	Number of row to be padded on bottom

3.16 Software Image Signal processor (Soft ISP)

This applet performs the following operations on a 16-bit RAW or Companded input image from an RCCC type sensor

- Decompanding
- Black Clamp and C Balance correction
- CFA Interpolation – RCCC to CCCC conversion
- Global Brithness and Contrast Enhancement (GBCE)
- Statistics Collection for Auto Exposure (AE)
- Extraction of R pixels

The decompanding, statistics collection and extraction of red (R) pixels are optional and can be enabled or disabled for each frame. The decompanding functionality can be used to decompand sensor inputs companded using a three segment piecewise linear transfer function. For GBCE, user can select between two methods – a simple GBCE and an interpolated look-up method. While the interpolated GBCE gives better quality output, the simple GBCE is faster.



3.16.1 Data Structures

3.16.1.1 SOFT_ISP_TI_CreateParams

Description

This structure defines the creation parameters for the Soft ISP applet

Fields

Field	Data Type	Input/ Output	Description
visionParams	IVISION_Params	Input	Commom structure for vision modules
imageWidth	uint16_t	Input	Width in pixels for the input image
imageHeight	uint16_t	Input	Height in number of lines for the input image
enableExtractR	uint8_t		Flag to indicate whether R pixels needs to be extracted or not.
enableStats	uint8_t	input	Flag to indicate whether Statistics Collector needs to be enabled for the current object or not.
statBlkWidth	uint16_t	input	Block width at which statistics needs to be collected.
statBlkHeight	uint16_t	input	Block height at which statistics needs to be collected.
minInputBufHeight	uint16_t	output	Minimum height expected for input buffer. The algorihtm will access the extra lines after valid imageHeight during processing, but the image content there will not effect the final output.
extraInputMem	uint16_t	output	Extra number of bytes required after the last valid input pixel (for a buffer size of imageWidth by imageHeight).
minOutputBufStride	uint16_t	output	Minimum output buffer stride (in bytes) expected to be provided to hold the output for the given input image dimension.
minOutputBufHeight	uint16_t	output	Minimum height of the output buffer to hold processed output for the given input image dimension.
statBufWidth	uint16_t	output	Width of the statistics buffer in bytes.
statBufHeight	uint16_t	output	Height of the statistics buffer
statBufStride	uint16_t	output	Minimum stride required for the statistics buffer.

3.16.1.2 SOFT ISP Applet Input and Output Buffer Indices

Description

The following buffer indices are used to refer to the various input and output buffers needed by the SOFT ISP applet.

Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
SOFT_ISP_TI_BUFDESC_IN_RCCC_IMAGE	enum	Input	This buffer descriptor (inBufs->bufDesc[SOFT_ISP_TI_BUFDESC_IN_RCCC_IMAGE])provides the input RCCC image data expected by applet. The input image is expected to be of 16-bit per pixel.
SOFT_ISP_TI_BUFDESC_OUT_CCCC_IMAGE	enum	Output	This buffer descriptor (outBufs->bufDesc [SOFT_ISP_TI_BUFDESC_OUT_CCCC_IMAGE]) should point to a buffer capable of holding the SOFT ISP output. The output is in 8-bit per pixel format.
SOFT_ISP_TI_BUFDESC_OUT_STATS_BUF		Output	This buffer descriptor should point to a buffer capable of holding statistics from the input image.
SOFT_ISP_TI_BUFDESC_OUT_R_IMAGE		Output	This buffer descriptor should point to a buffer for holding the R pixels extracted from the input frame. The output R pixels values will be of 8-bits per pixel.

3.16.1.3 SOFT ISP Applet Control Command Indices

Description

This following Control Command indices are supported for Soft ISP applet.
Fields

Field	Description
TI_SOFT_ISP_CONTROL_GET_BUF_SIZE	The algorithm expects the input and output buffers to be satisfy certain contraints. These details are communicated to the user using a control call with this index. Given the input create time parameters, the control call return the buffer constraints through the output parameters of SOFT_ISP_TI_CreateParams structure.

3.16.1.4 SOFT_ISP_TI_InArgs

Description

This structure contains all the parameters which are given as an output by SOFT ISP applet.
Fields

Field	Data Type	Input/ Output	Description
iVisionInArgs	IVISION_InArgs	Input	Common inArgs for all ivison based modules

Field	Data Type	Input/ Output	Description
`updateToneCurve`	`uint8_t`	Input	This flag indicates whether the GBCE tone curve needs to be updated for the current process call. If the flag is set to 1, the tone curve at pGbceToneCurve will be used for brightness and contrast enhancement. Else, the tone curve from previous frame will be reused. Note that for the very first frame this flag needs to be set to 1. Also the flag needs to be set whenever EVE is shared by any other applet or algorithm (in between any two process calls of soft ISP) even if the GBCE tone curve has not changed.
`pGbceToneCurve`	`uint8_t*`	Input	This points to the buffer containing the tone curve to be used for Global Brightness and Contrast Enhancement (GBCE). The tone curve needs to be of size 4*4096 bytes. The original tone curve is of 4096 bytes and the factor 4 indicates the replication required by EVE.
`sensorBitDepth`	`uint8_t`	Input	Maximum number of bits per input pixel. Currently only a bit depth of 12-bits is supported. (GBCE stage assumes this.)
`rPosition`	`uint8_t`	input	Location of the R pixel in the RCCC paxel. The location needs to be specified w.r.t the pixel at (frameROI.topLeft.x, frameROI.topLeft.y). Possible values are 1, 2, 3 and 4. 1 stands for RCCC, 2 for CRCC, 3 for CCRC and 4 for CCCR.
`enableDecompand`	`uint8_t`	input	Flag to indicate whether sensor data needs to be decompanded or not for the current frame. The decompand the sensor data is supported only if the companding is piece-wise linear with 3 segments (or 2 knee-points).
`pout1`	`uint16_t`	input	Sensor output at first knee-point in the piece-wise linear response.
`pout2`	`uint16_t`	input	Sensor output at second knee-point in the piece-wise linear response.
`slope1`	`uint8_t`	input	Slope of the decompanding piece-wise linear curve between the two knee points.
`slope2`	`uint16_t`	input	Slope of the decompanding piece-wise linear respose after the second knee point.
`blackClamp`	`uint16_t`	Input	The dark current values to be subtracted from each paxel. This is an array of 4 values. The four values correspond to the 4 locations in a paxel.
`cBalanceGain`	`uint16_t`	Input	Gain to be applied to each of the 4 pixels in a paxel. An array of 4 16-bit values is expected. The value to be programmed is equal to the actual gain multiplied by 2^(cBalanceShift).
`cBalanceShift`	`uint8_t`	Input	The shift (right shift) required to scale down the cBalanceGain values.

Field	Data Type	Input/ Output	Description
`enableExtractR`		Input	Flag to indicate whether R pixels needs to be extracted or not for the current frame. This will have effect only if enableExtractR was enabled during create time.
`gbceMethod`		Input	This indicates the method to be employed for GBCE computation. There are two supported methods - GBCE simple and GBCE interpolated. In GBCE simple the pixel value, if more than 12 bits, is truncated to 12-bit before looking up in the LUT. In the interpolated GBCE method, the output is the result of bilinear interpolation from the two nearby entries in the 12-bit LUT. The value 0 stands for simple GBCE and 1 for interpolated GBCE.
`enableStats`		Input	Flag to indicate whether Statistics Collector needs to be enabled for the current frame or not. This will have eeffect only if enableStats was enabled during create time.
`saturationLimit`		Input	Pixel value beyond which the pixel will be considered as saturated during statistics collection.

The GBCE tone curve maps 12-bit input to an 8-bit output. Such a tone curve would have 4096 entires. The tone curve that needs to be send to the applet needs to be replicated 4 times. The resulting tone curve will have 4*4096 entires.

A group of eight entries needs to be repeated 4 times as shown below:

tc[0] tc[1] tc[2] tc[3] tc[4] tc[5] tc[6] tc[7] tc[0] tc[1] tc[2] tc[3] tc[4] tc[5] tc[6] tc[7] tc[0] tc[1] tc[2] tc[3] tc[4] tc[5] tc[6] tc[7] tc[0] tc[1] tc[2] tc[3] tc[4] tc[5] tc[6] tc[7]

tc[8] tc[9] tc[10] tc[11] t[12] … tc[15] tc[8] tc[9] tc[10] tc[11] t[12] … tc[15] tc[8] tc[9] tc[10] tc[11] t[12] … tc[15] tc[8] tc[9] tc[10] tc[11] t[12] … tc[15]

: : : :

3.16.1.5 *SOFT_ISP_TI_OutArgs*

Description

This structure contains all the parameters which are given as an output by SOFT ISP applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules

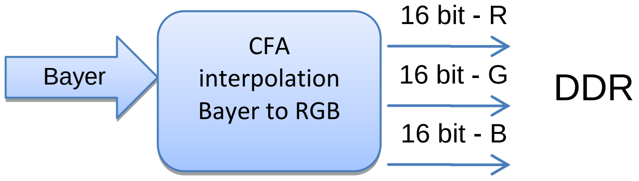
3.16.2 Constraints

- Input image pixel should be of 16-bit resolution with actual bit-depths varying from 12-bit to 16-bit. In case of companded inputs, the restriction is only on the companded pixel. The decompanded pixel bit-depth could be more than 16-bit.
- Image width and Image Height should be a multiple of 2.
- Statistics block width has to be a multiple of 16 and statistics block height has to be a multiple of 2.
- Depending on the input image width and image height, the input buffer might be expected to have additional memory or pixels at the end of the buffer. The additional memory requirement is notified to the user as the minimum input image height expected and additional bytes required apart from the minimum height.
- Statistics at the boundary region of the image (right and bottom boaundaries) may not be valid. This is indicated to user via the statBufStride parameter during the control call for getting the buffer sizes. In case boundary data is invalid statBufWidth will be lesser than the statBufStride.
- The applet can handle only statistics block size that can be handled within the limited internal memory. During create time this is indicated via the error code ISOFT_ISP_ERRORTYPE_STAT_BLK_TOO_BIG.

3.17 Software Image Signal processor (Soft ISP) for Bayer sensor

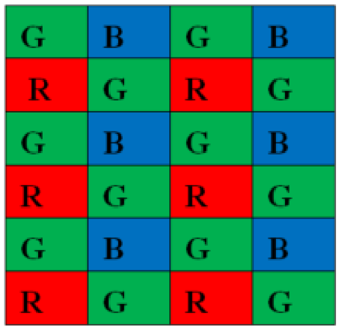
This applet performs the following operations on a 16-bit RAW Bayer type sensor

- CFA Interpolation – Bayer to RGB 16-bits

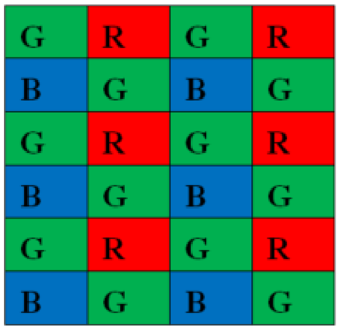


Note that although this ISP only includes the CFA interpolation, it can be easily extended to include other common image processing module such as white balance, tone mapping, color correction, edge enhancement, etc.

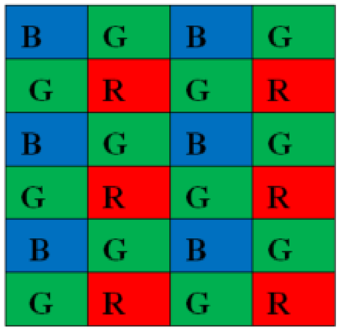
The applet can handle all 4 bayer patterns:



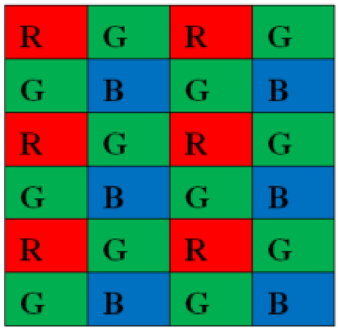
'gbrg' pattern



'grbg' pattern



'bggr' pattern



'rggb' pattern

Note that the input image must be prepadded with a border of 3 pixels wide on top, bottom, left and right of the region of interest or include valid pixels.

3.17.1 Data Structures

3.17.1.1 SOFT_ISP16_TI_CreateParams

Description

This structure defines the creation parameters for the Soft ISP applet

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`imageWidth`	`uint16_t`	Input	Width in pixels for the input image
`imageHeight`	`uint16_t`	Input	Height in number of lines for the input image
`bayerPattern`	`int32_t`	Input	Enum indicating the bayer pattern passed as input: `SOFT_ISP16_BAYER_PATTERN_GBRG`, `SOFT_ISP16_BAYER_PATTERN_GRBG`, `SOFT_ISP16_BAYER_PATTERN_BGGR`, `SOFT_ISP16_BAYER_PATTERN_RGGB`

3.17.1.2 SOFT ISP16 Applet Input and Output Buffer Indices

Description

The following buffer indices are used to refer to the various input and output buffers needed by the SOFT ISP16 applet.

Fields

Field	Data Type	Input/ Output	Description
`SOFT_ISP16_TI_BUFDESC_IN`	`enum`	Input	This buffer descriptor (inBufs->bufDesc[SOFT_ISP16_TI_BUFDESC_IN]) provides the input Bayer image data expected by applet. The input image is expected to be of 16-bit per pixel.
`SOFT_ISP16_TI_BUFDESC_OUT`	`enum`	Output	This buffer descriptor (outBufs->bufDesc [SOFT_ISP16_TI_BUFDESC_OUT]) should point to a 3-planes buffer capable of holding the SOFT ISP output for R, G, B planes. To initialize each plane’s pointer, the calling application must set outBufs->bufDesc [SOFT_ISP16_TI_BUFDESC_OUT].numPlanes to 3 and fill the structure outBufs->bufDesc [SOFT_ISP16_TI_BUFDESC_OUT.bufPlanes[i] for i=0, 1, 2.

3.17.1.3 SOFT ISP16 Applet Control Command Indices

Description

This following Control Command indices are supported for Soft ISP applet.

Fields

Field	Description
`TI_SOFT_ISP16_CONTROL_GET_BUF_SIZE`	The algorithm expects the input and output buffers to be satisfy certain contraints. These details are communicated to the user using a control call with this index. Given the input create time parameters, the control call return the buffer constraints through the output parameters of SOFT_ISP_TI_CreateParams structure.

3.17.1.4 *SOFT_ISP16_TI_InArgs*

Description

This structure contains all the parameters which are given as an output by SOFT ISP16 applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_InArgs`	Input	Common inArgs for all ivison based modules

3.17.1.5 *SOFT_ISP16_TI_OutArgs*

Description

This structure contains all the parameters which are given as an output by SOFT ISP applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules
`activeImageWidth`	`uint16_t`	Output	Width in bytes of the area that will be accessed by the EDMA when reading the input frame. For this function, it should always be equal to 2 *(imgFrameWidth + 6) bytes .
`activeImageHeight`	`uint16_t`	Output	Height in lines of the area that will be accessed by the EDMA when reading the input frame. For this function, it should always be equal to (imgFrameHeight + 6) lines .
`outputBlockWidth`	`uint16_t`	Output	The output frame will be written block-wise. Each block will be of dimension outputBlockWidth x outputBlockHeight. The value of outputBlockWidth is returned in this parameter and will be a divisor of imgFrameWidth.
`outputBlockHeight`	`uint16_t`	Output	The output frame will be written block-wise. Each block will be of dimension outputBlockWidth x outputBlockHeight. The value of outputBlockHeight is returned in this parameter and will be a divisor of imgFrameHeight.

3.17.2 Constraints

- Input image pixel should be of 16-bit resolution with actual bit-depths varying from 12-bit to 16-bit.
- Image width must be a multiple of 16 and Image height must be a multiple of 2.

3.18 Hough Transform for lines

This applet calculates the hough transform for a given set of edge points. This routine accepts an input list of edges in packed for with respect to x and y with each x and y being 16 bit quantity. It also accepts the input image dimesnsion along with the range of rho and theta for which transform needs to be calculated.

3.18.1 Data Structures

3.18.1.1 HOUGH_FOR_LINES_TI_CreateParams

Description

This structure defines the creation parameters for the Hough For lines applet

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules

3.18.1.2 IHOUGH_FOR_LINES_InBufOrder

Description

User provides most of the infomration through buffer descriptor during process call.

The below enums define the purpose of the input buffer.

Fields

Enumeration	Description
`HOUGH_FOR_LINES_TI_BUFDESC_IN_XY_LIST`	This buffer descriptor (inBufs->bufDesc[HOUGH_FOR_LINES_TI_BUFDESC_IN_XY_LIST]) should point to a buf descriptor containing XY list of the edge points in an image. This list is expected to be in packed 32 bit format with X coordinate followed by Y coordinate for each edge point. Each X or Y is a 16 bit entry..
`HOUGH_FOR_LINES_TI_BUFDESC_IN_TOTAL`	This indicates total number of input buffers expected.

3.18.1.3 IHOUGH_FOR_LINES_OutBufOrder

Description

User provides most of the infomration through buffer descriptor during process call.

Below enums define the purpose of out buffer.

Fields

Enumeration	Description

Enumeration	Description
HOUGH_FOR_LINES_TI_BUFDESC_OUT_RHO_THETA_SPACE	his buffer descriptor (outBufs->bufDesc[HOUGH_FOR_LINES_TI_BUFDESC_OUT_RHO_THETA_SPACE]) should point to the output buffer which will contain the list of rho and theta points for all the edge points by this applet. Each rho and theta is a 16 bit entry packed in 32 bit format with rho followed by theta.
HOUGH_FOR_LINES_TI_BUFDESC_OUT_TOTAL	This indicates total number of output buffers expected.

3.18.1.4 HOUGH_FOR_LINES_InArgs

Description

This structure contains all the parameters which are given as an output by Hough For Lines applet.

Fields

Field	Data Type	Input/ Output	Description
iVisionInArgs	IVISION_InArgs	Input	Common inArgs for all ivison based modules
listSize	uint16_t	Input	Size of the list of edge points in terms of number of points.
thetaStart	uint16_t	Input	Starting theta in degrees for the region in which lines are supposed to be detected.
thetaEnd	uint16_t	Input	End theta in degrees for the region in which lines are supposed to be detected
thetaStepSize	uint8_t	input	Steps in which theta should be incremented while moving from thetaMin to thetaMax.
rhoMaxLength	uint16_t	input	Maximum rho value in rho theta space
imgWidth	uint16_t	input	Image frame width.
imgHeight	uint16_t	input	Image frame height

: :

3.18.1.5 HOUGH_FOR_LINES_OutArgs

Description

This structure contains all the parameters which are given as an output by Hough For Lines applet.

Fields

Field	Data Type	Input/ Output	Description
iVisionOutArgs	IVISION_Params	Output	Commom structure for outArgs ivision modules

3.18.2 Constraints

- Theta Values should lie between 0 to 360 degrees
- Rho Max value should be less than HOUGH_FOR_LINES_TI_MAXRHOLENGTH
- List Size should be less than HOUGH_FOR_LINES_TI_MAXLISTSIZE.

3.19 Integral Image

This routine computes the integral image of an unsigned 8-bits image. The output data is in 32-bit unsigned format allowing an input image of up to 16 Mpix without causing any overflow.

3.19.1 Data Structures

3.19.1.1 INTEGRAL_IMAGE_TI_CreateParams

Description

This structure defines the creation parameters for integral image applet.

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`imageFrameWidth`	`uint16_t`	Input	Width of the input image
`imageFrameHeight`	`uint16_t`	Input	Height of the input image

3.19.1.2 INTEGRAL_IMAGE_TI_OutArgs

Description

This structure contains all the parameters which are given as an output by integral image applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules
`blockWidth`	`uint16_t`	Output	The input frame is partitioned into blocks of blockWidth x blockHeight that are processed one after the other by the VCOP. The value of blockWidth is returned in this parameter.
`blockHeight`	`uint16_t`	Output	The input frame is partitioned into blocks of blockWidth x blockHeight that are processed one after the other by the VCOP. The value of blockHeight is returned in this parameter.

3.19.2 Constraints

- imageFrameWidth must be multiple of 16.
- imageFrameHeight must be multiple of 8.
- imageFrameWifth x imageFrameHeight must be <= 16,777,216 pixels to avoid overflow.

3.20 NMS (Non Maximum Suppression)

This applet performs Non Maximum Suppression (NMS) along with thresholding on the input data given by the user This routine accepts an input data and output a list of x and y coordinates in packed format of x and y.

3.20.1 Data Structures

3.20.1.1 NMS_TI_CreateParams

Description

This structure defines the creation parameters for the NMS applet

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`inputByteDepth`	`uint8_t`	Input	Describes the bit depth for the input data. It can be 2 or 4 bytes. Refer INMS_InputByteDepth for valid enteries

3.20.1.2 NMS_TI_InBufOrder

Description

User provides most of the infomration through buffer descriptor during process call.

The below enums define the purpose of the input buffer.

Fields

Enumeration	Description
`NMS_TI_BUFDESC_IN_IMAGEBUFFER`	This buffer descriptor (inBufs->bufDesc[NMS_TI_BUFDESC_IN_IMAGEBUFFER]) should point to a buf descriptor pointing to input image data. Input image width and height should be multiple of 8.
`NMS_TI_BUFDESC_IN_TOTAL`	This indicates total number of input buffers expected.

3.20.1.3 NMS_TI_OutBufOrder

Description

User provides most of the infomration through buffer descriptor during process call.

Below enums define the purpose of out buffer.

Fields

Enumeration	Description

Enumeration	Description
`NMS_TI_BUFDESC_OUT_LIST_XY`	This buffer descriptor (outBufs->bufDesc[NMS_TI_BUFDESC_OUT_LIST_XY]) should point to the output buffer which will contain XY list of the after NMS with thresholding. This list is expected to be in packed 32 bit format with X coordinate followed by Y coordinate for each edge point. Each X or Y is a 16 bit entry
`NMS_TI_BUFDESC_OUT_TOTAL`	This indicates total number of output buffers expected.

3.20.1.4 NMS_TI_ControlInArgs

Description

This structure contains all the parameters needed for control call of this applet.

Fields

Field	Data Type	Input/ Output	Description
`imageBuf`	`IVISION_InBufs`	Input	This is the input buffer which is provided in process call. Control function will determine the region for which this kernel would be running for the input dimensions givenby the user. User is expected to allocate this much memory for inout buffer
`windowWidth`	`uint8_t`	Input	Width of the window for NMS
`windowHeight`	`uint8_t`	Input	Height of the window for NMS

3.20.1.5 NMS_TI_ControlOutArgs

Description

This structure contains all the parameters that are returned by control call of this applet.

Fields

Field	Data Type	Input/ Output	Description
`effectiveImageWidth`	`Uint16_t`	Output	effective image width for which this applet is working. User should atleast allocate this much memory
`effectiveImageHeight`	`Uint16_t`	Output	effective image height for which this applet is working. User should atleast allocate this much memory

3.20.1.6 NMS_TI_InArgs

Description

This structure contains all the parameters which are given as an output by NMS applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_InArgs`	Input	Common inArgs for all ivison based modules
`nmsThreshold`	`Int32_t`	Input	Threshold to be used with NMS.
`windowWidth`	`uint8_t`	Input	Width of the window for NMS
`windowHeight`	`uint8_t`	Input	Height of the window for NMS

: :

3.20.1.7 NMS_TI_OutArgs

Description

This structure contains all the parameters which are given as an output by NMS applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules
`numListPoints`	`uint32_t`	Output	Number of points in the list after NMS and thresholding

3.20.2 Constraints

- Width and Height should be multiple of 8
- Maximum width supported is 1920

3.21 Census transform

Overview

This applet computes the census transform of a 8-bits or 16-bits input frame. The function is generic enough to accept any dimensions (winWidth x winHeight) for the support window. A support window is the rectangle delimiting the neighbourhood within which each pixel is compared with the center pixel to produce a census codeword.
A census codeword is a bit string, composed of the boolean comparisons of the center pixel with each of its neighborhood inside the support window. A bit '1' indicates that the center pixel's value is greater or equal than its neighbor's value, '0' indicates the opposite. The bit string is produced in a raster scan format with bit #0 corresponding to the comparison with the upper left neighbor and the last bit corresponding to the comparison with the lower right neighbor.
In theory, the bit string length should be windWidth * windHeight bits (note that center pixel comparison is included). But bit string lengths that are not multiple of 8, are hard to manipulate because they don't round up to an even number of byte and would need concatenation for post processing. Thus the function produces a more friendly representation by rounding up the bit string to a byte boundary. To achieve this, bits of value 0 are inserted at the end of each bit string until a byte boundary is reached.

For example, let's have this 3x3 pattern:

0 0 0
1 1 1
1 2 0

Applying a 3x3 census transform to the center pixel would produce the following bit string:

Bit #0	Bit #1	Bit #2	Bit #3	Bit #4	Bit #5	Bit #6	Bit #7	Bit #8	Bit #9	Bit #10	Bit #11	Bit #12	Bit #13	Bit #14	Bit #15
1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0

Note that bits #9 to #15 are 0 padding bits inserted to make the bit string align with a byte boundary.

Speed-up with downsampling

It is often desirable to keep a codeword length small in order to speed up processing and keep the amount of memory to store the output low. For instance a 15x15 census window requires floor((15*15+7)/8)= 29 bytes per codeword. The resulting census frame will be 29x larger than the input frame assuming 8-bits input element. Census transform outputs are generally fed to some feature matching algorithm relying on hamming distance and codeword sizes of 1, 2, 4 bytes are more manageable as they typically correspond to a processor's data bandwidth. So a codeword size of 29 bytes is certainly too big to be efficiently handled. As a mean to limit codeword's size, this function offers the option to downsample a support window in the horizontal or vertical direction through the setting of parameters winHorzStep and winVertStep. The downside of downsampling is of course less information captured in the codeword. Since neighbor pixels are usually highly correlated, the loss of information may be a small, whereas the benefit in processing speed-up and memory saving is quiet significant.

To illustrate the concept, let's have a 5x5 support window:

o		o		o
o		X		o
o		o		o

If winHorzStep= winVertStep= 1, the center pixel 'X' would have to be compared with each of its 25 neighbors, producing a 4 bytes codeword (25 bits aligned to a byte boundary). But if winHorzStep= winVertStep= 2, then only the neighbors represented by a 'o' are used to generate a codeword, which will be 2 bytes long.

Number of bytes per census codeword

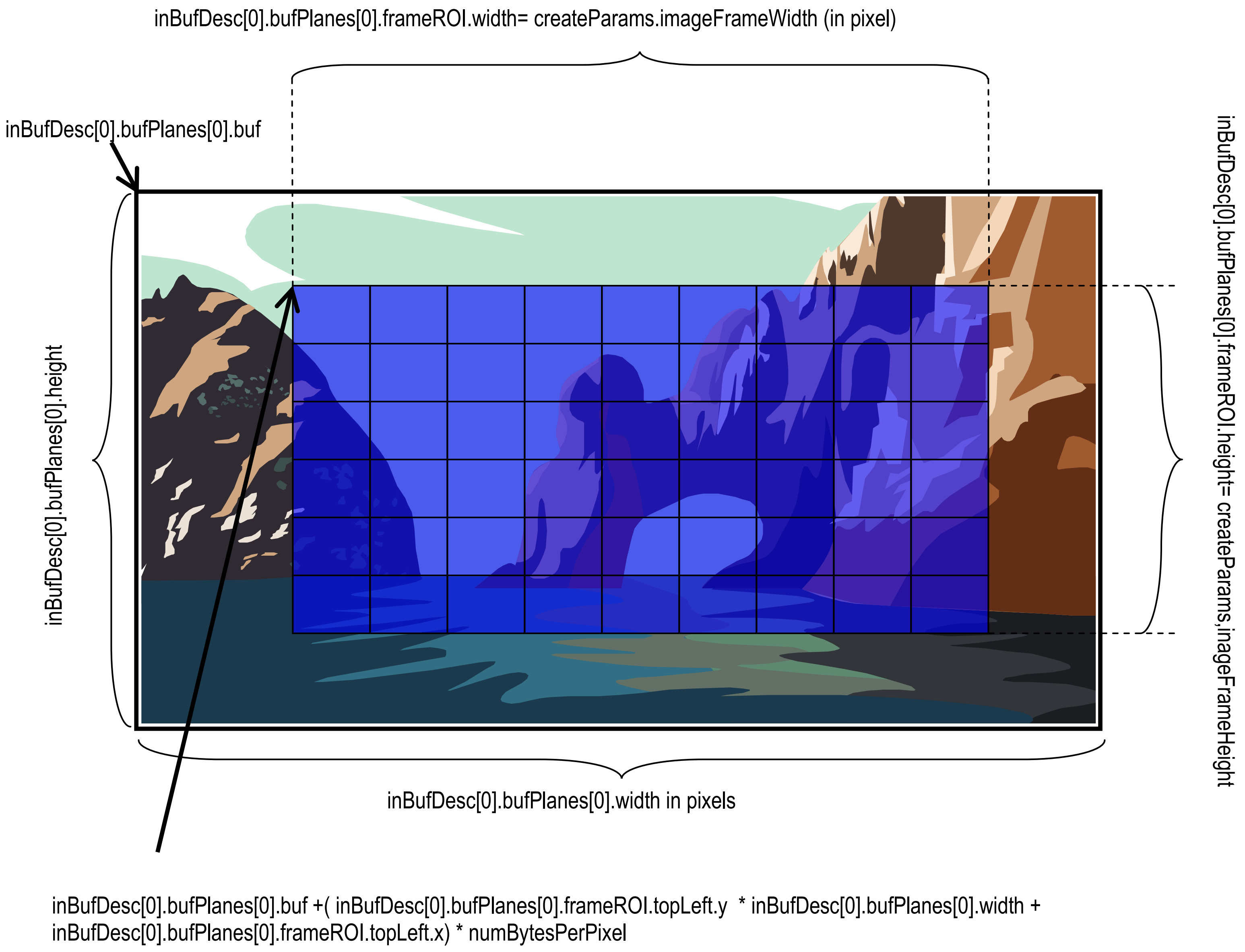
In conclusion, the formula that gives the length of each bit string, in number of bytes is as follow:

numBytesPerCensus= (((winWidth + winHorzStep -1) / winHorzStep)*((winHeight + winVertStep -1) / winVertStep) + 7)/8

So for a 15x15 census window and winHorzStep=2, winVertStep=2, a codeword length would be 8 bytes. For output buffer user should allocate imageFrameWidth * imageFrameHeight * numBytesPerCensus.

Frame layout

The figure below shows to what dimensions the different members of the CENSUS_TI_CreateParams and IVISION_BufDesc correspond to with respect to the input frame.


inBufDesc[0].bufPlanes[0].frameROI.width= createParams.imageFrameWidth (in pixel)
inBufDesc[0].bufPlanes[0].buf
inBufDesc[0].bufPlanes[0].height
inBufDesc[0].bufPlanes[0].frameROI.height= createParams.imageFrameHeight
inBufDesc[0].bufPlanes[0].width in pixels
inBufDesc[0].bufPlanes[0].buf +(inBufDesc[0].bufPlanes[0].frameROI.topLeft.y * inBufDesc[0].bufPlanes[0].width + inBufDesc[0].bufPlanes[0].frameROI.topLeft.x) * numBytesPerPixel

At creation time, the members createParams.imageFrameWidth and createParams.imageFrameHeight are passed through the creation params structure CENSUS_TI_CreateParams. These values correspond to the actual ROI's width and height, depicted by the blue region in the figure above, which will be processed by algProccess(). Using the values, the underlying creation function (initiated by algInit) will allocate the onchip memory accordingly and partition the frame in blocks. The dimensions of the blocks can be obtained after creation, by calling the algControl() function, which will return controlOutParams->outputBlockWidth and controlOutParams->outputBlockHeight. Note that for this function, the ROI's width and height can be changed after creation since the ROI dimensions are passed to `inBufDesc[0].bufPlanes[0].frameROI.width` and `inBufDesc[0].bufPlanes[0].frameROI.height`, which are passed to algProcess(). Normally for other applets, the inBufDesc[]…frameROI.width and inBufDesc[]….frameROI.height values must exactly match the values used in createParams.imageFrameWidth and createParams.imageFrameHeight. In other words, once the ROI width and height are set at create time, it is not possible to use different values at process time without causing errors. For this applet though, it is possible to re-specify these values by just setting inBufDesc[0].bufPlanes[0].frameROI.width and inBufDesc[0].bufPlanes[0].frameROI.height with new ones. The only constraint is that these new dimensions must be multiple of controlOutParams->outputBlockWidth and controlOutParams->outputBlockHeight, which are returned by algControl() .

3.21.1 Data Structures

3.21.1.1 CENSUS_TI_CreateParams

Description

This structure defines the creation parameters for census transform applet.

Fields

Field	**Data Type**	**Input/ Output**	**Description**
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`imageFrameWidth`	`uint16_t`	Input	Width of the input image's ROI
`imageFrameHeight`	`uint16_t`	Input	Height of the input image,s ROI
`inputBitDepth`	`uint8_t`	Input	Bit depth of the input image (8 or 16)
`winWidth`	`uint8_t`	Input	Width of the support window
`winHeight`	`uint8_t`	Input	Height of the support window
`winHorzStep`	`uint8_t`	Input	Horizontal step between each position in the support window
`winVertStep`	`uint8_t`	Input	Vertical step between each position in the support window

3.21.1.2 CENSUS_TI_OutArgs

Description

This structure contains all the parameters which are produced as an output by the census transform applet.

Fields

Field	**Data Type**	**Input/ Output**	**Description**
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules
`activeImageWidth`	`uint16_t`	Output	Width in bytes of the area that will be accessed by the EDMA when reading the input frame. For this function, it should always be equal to numBytesPerPixel *(imgFrameWidth + winWidth – 1) bytes .

Field	Data Type	Input/ Output	Description
`activeImageHeight`	`uint16_t`	Output	Height in lines of the area that will be accessed by the EDMA when reading the input frame. For this function, it should always be equal to (imgFrameHeight + winHeight – 1) lines .
`outputBlockWidth`	`uint16_t`	Output	The output frame will be written block-wise. Each block will be of dimension outputBlockWidth x outputBlockHeight. The value of outputBlockWidth is returned in this parameter and will be a divisor of imgFrameWidth.
`outputBlockHeight`	`uint16_t`	Output	The output frame will be written block-wise. Each block will be of dimension outputBlockWidth x outputBlockHeight. The value of outputBlockHeight is returned in this parameter and will be a divisor of imgFrameHeight.
`numBytesPerCensus`	`uint8_t`	Output	Census transform codeword's length in bytes

3.21.1.3 *CENSUS_TI_ControlOutParams*

Description

This structure contains all the parameters which are produced as an output by the census transform applet's algControl() function.

Fields

Field	Data Type	Input/ Output	Description
`algParams`	`IALG_Params`	Output	Commom structure for algParams ivision modules
`outputBlockWidth`	`uint16_t`	Output	The output frame will be written block-wise. Each block will be of dimension outputBlockWidth x outputBlockHeight. The value of outputBlockWidth is returned in this parameter and will be a divisor of imgFrameWidth.
`outputBlockHeight`	`uint16_t`	Output	The output frame will be written block-wise. Each block will be of dimension outputBlockWidth x outputBlockHeight. The value of outputBlockHeight is returned in this parameter and will be a divisor of imgFrameHeight.

3.21.2 Constraints

- imageFrameWidth must be multiple of 16.
- imageFrameHeight must be multiple of 8.
- inBufDesc[0].bufPlanes[0].width >= inBufDesc[0].bufPlanes[0].frameROI.width + createParams.winWidth - 1
- inBufDesc[0].bufPlanes[0].height >= inBufDesc[0].bufPlanes[0].frameROI.height + createParams.winHeight - 1
- inBufDesc[0].bufPlanes[0].frameROI.width must be multiple of controlOutParams->outputBlockWidth
- inBufDesc[0].bufPlanes[0].frameROI.height must be multiple of controlOutParams->outputBlockHeight

- inBufDesc[0].bufPlanes[0].frameROI.topLeft.x >= (createParams.winWidth – 1)/2 (to ensure correct values in left and bottom band)
- inBufDesc[0].bufPlanes[0].frameROI.topLeft.y >= (createParams.winHeight – 1)/2 (to ensure correct values in top and bottom band).
- inputBitDepth must be either 8 or 16.

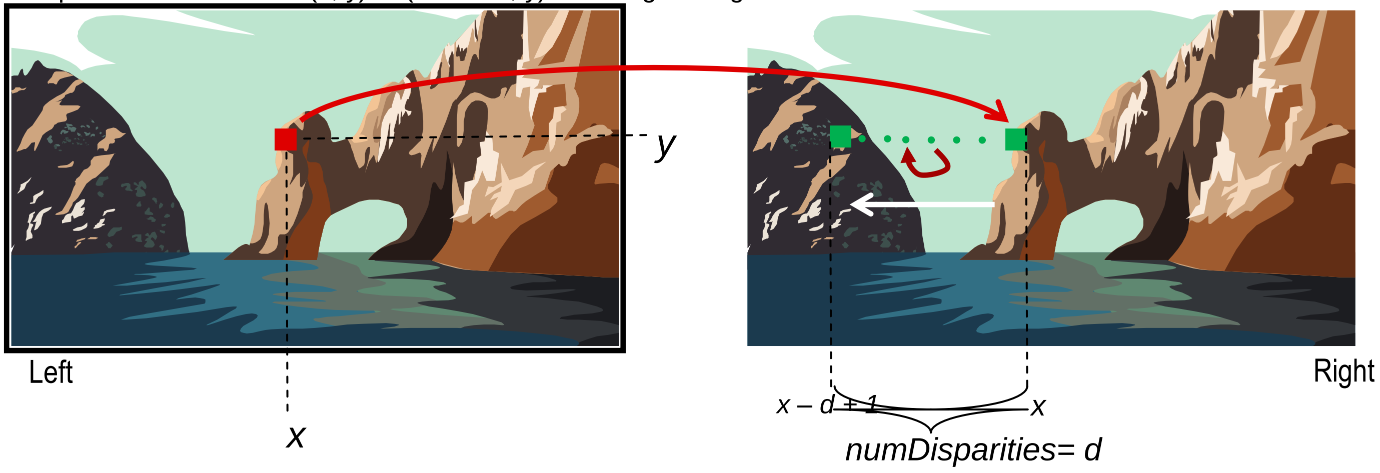
3.22 Disparity calculation

Overview

This applet computes the disparity map for stereovision between two images. In order to respect the epipolar constraint, it is assumed that the two images are from calibrated cameras or have been rectified, so that they are vertically aligned in order for the horizontal lines of pixels to coincide. This way, the underlying correspondence search is a 1-D problem instead of a 2-D problem.

The disparity algorithm uses horizontal block matching as correspondence matching. The correspondence matching is carried out by matching one image with the other. There are two modes of matching direction: left to right or right to left. The user of the function can set `createParams.searchDir` to either `DISPARITY_TI_RIGHT_TO_LEFT or DISPARITY_TI_LEFT_TO_RIGHT` to select between the two modes.

When `DISPARITY_TI_LEFT_TO_RIGHT` is selected, each pixel's neighborhood in the left image is compared with many other neighborhood positions in the right image along the same epipolar line. A neighborhood is defined as a rectangle of size winWidth x winHeight in which a cost value is calculated and aggregated. This process is called block matching along epi-polar line. The figure below shows a pixel in the left image at coordinates (x,y) being compared with pixels at coordinates (x, y) to (x – d +1, y) in the right image.



When `DISPARITY_TI_RIGHT_TO_LEFT` is selected, each pixel's neighborhood in the right image is compared with many other neighborhood positions in the left image along the same epipolar line.

In theory, either mode should produce the same output for the center region of the image that excludes left and right bands of width numDisparities – 1 pixels. However due to occlusions or flat textures, the results may be different. That's why in some use cases, it is desirable to run the function along one search direction and then again along the opposite direction. Then, post-processing can discard the output locations for which the disparity obtained using one dirrection is too different from the other direction.

During block matching, the cost can be calculated either by using SAD (sum of absolute difference) if the input images are intensity images or by hamming distance if the imput images are made up of census codewords. Current implementation only supports hamming distance,

The output disparity map is a 8-bit frame in which each value corresponds to the displacement associated to the minimum cost found during the horizontal block matching.

To restrict the search range, the applet accepts as parameter the number of disparities to search, numDisparities, which typically ranges from 32 to 128. The smaller the number of disparities, the faster the execution time is. However limiting the range of disparities to a small number might produce erroneous disparities for near objects. One way to cope with this is to keep a wide range of disparities while reducing the number of disparities to search. This can be done by setting the disparity step to some value greater than 1. The effect will be that the disparity range will be downsampled, resulting in fewer disparities to calculate.

For instance, if numDisparities= 128 and disparityStep= 4, then only disparities 0,4,8,…,120 will be searched and the computational cost will be the same as numDisparities= 32 and disparityStep= 1. Of course, the quality of the disparity map will be degraded due to quantization effect so one needs to carefully consider this side effect, which can be mitigated with interpolation in a later post-processing stage.

In addition to the disparity map, the applet can also output the minimum cost values but also values adjacent to the minimum costs for the purpose of sub-pixel disparity estimation by parabola interpolation method, which can be handled by a separate post-processing step.

Frame layout

The information in this section is complemented by the document *apps/disparity/docs/disparity_parameters.pdf.*

The figure below shows to what dimensions the different members of the DISPARITY_TI_CreateParams and IVISION_BufDesc correspond to with respect to the input left or right frame.

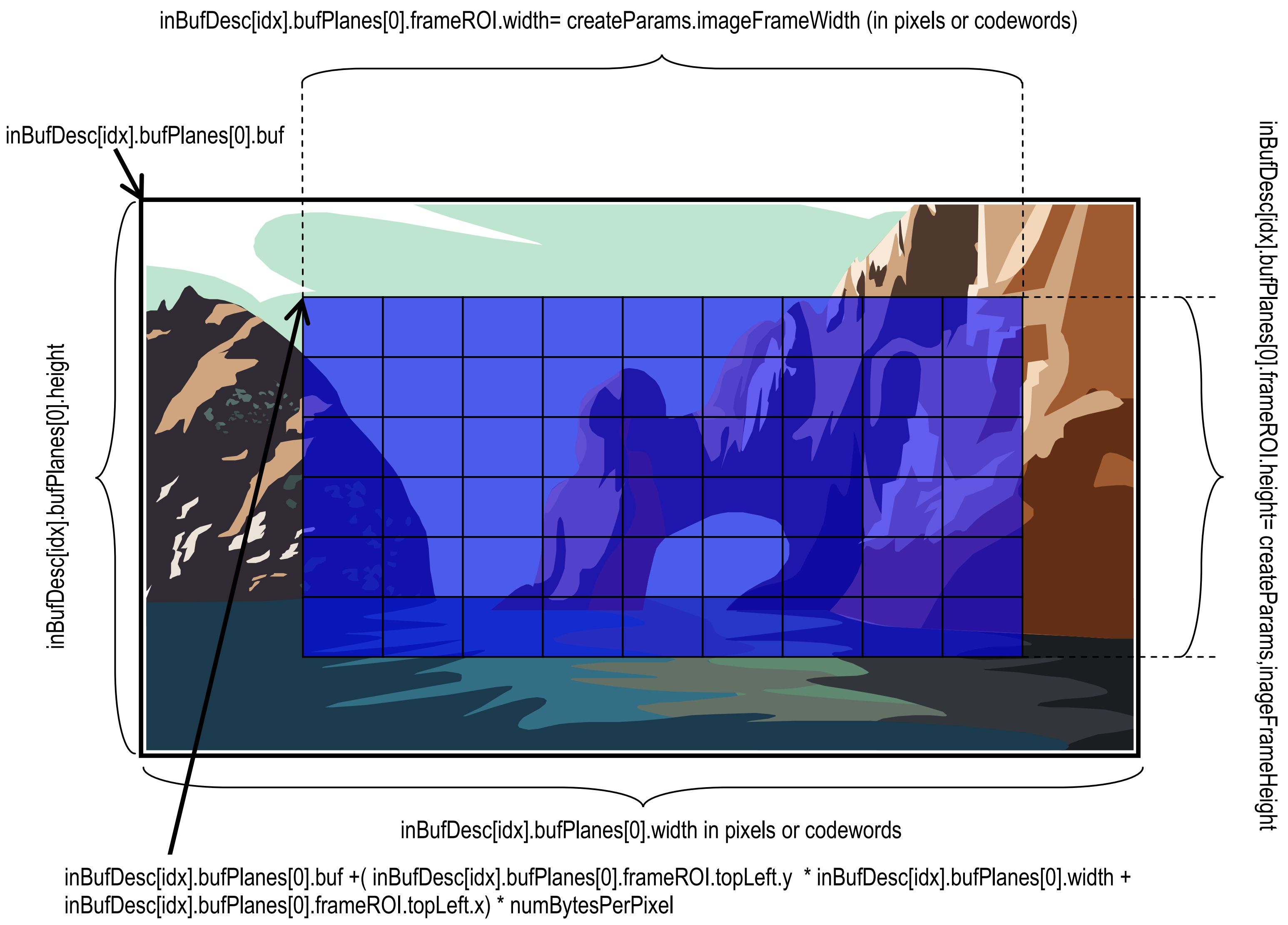

inBufDesc[idx].bufPlanes[0].frameROI.width= createParams.imageFrameWidth (in pixels or codewords)
inBufDesc[idx].bufPlanes[0].buf
inBufDesc[idx].bufPlanes[0].height
inBufDesc[idx].bufPlanes[0].frameROI.height= createParams.imageFrameHeight
inBufDesc[idx].bufPlanes[0].width in pixels or codewords
inBufDesc[idx].bufPlanes[0].buf +(inBufDesc[idx].bufPlanes[0].frameROI.topLeft.y * inBufDesc[idx].bufPlanes[0].width +
inBufDesc[idx].bufPlanes[0].frameROI.topLeft.x) * numBytesPerPixel

Figure 1. Frame layout of the disparity applet

For left frame, idx= `DISPARITY_TI_BUFDESC_IN_LEFT_IMAGE` (abreviated to `LEFT` in the remaining document) and for right image, idx= `DISPARITY_TI_BUFDESC_IN_RIGHT_IMAGE` (abreviated to `RIGHT` in the remaining document) and for right image.

As depicted by the above layout, the following constraints should be respected in order to avoid producing invalid disparity outputs.

`inBufDesc[LEFT].bufPlanes[0].width`	≥	`inBufDesc[LEFT].bufPlanes[0].frameROI.width + createParams.winWidth - 1 +` ***`createParams.numDisparities - 1`***

inBufDesc[LEFT].bufPlanes[0].height	≥	inBufDesc[LEFT].bufPlanes[0].frameROI.height + createParams.winHeight -1
inBufDesc[RIGHT].bufPlanes[0].width	≥	inBufDesc[RIGHT].bufPlanes[0].frameROI.width + createParams.winWidth – 1 + ***createParams.numDisparities – 1***
inBufDesc[RIGHT].bufPlanes[0].height	≥	inBufDesc[RIGHT].bufPlanes[0].frameROI.height + createParams.winHeight -1

Note that `frameROI.width` and `frameROI.height` values are the same between left and right frames. If `createParams.searchDir= DISPARITY_TI_RIGHT_TO_LEFT`, the extra term `createParams.numDisparities – 1` present in the `'width'` constraints may be optional. If omitted, some disparities in the right-hand side of the image will be invalid. If `createParams.searchDir= DISPARITY_TI_LEFT_TO_RIGHT`, this extra term must be present.

If `createParams.searchDir= DISPARITY_TI_RIGHT_TO_LEFT` is selected and if desired output disparity range is `[0, numDisparities -1 ]` then additional constraints are:

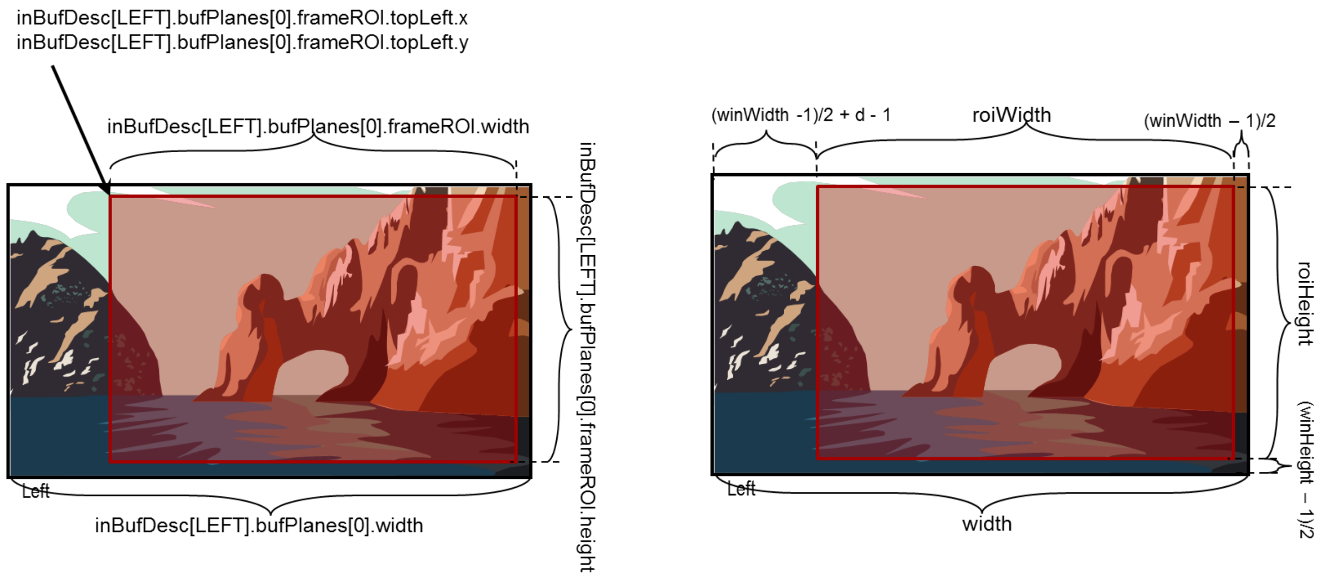
inBufDesc[LEFT].bufPlanes[0].frameROI.topLeft.x	≥	(createParams.winWidth – 1)/2
inBufDesc[LEFT].bufPlanes[0].frameROI.topLeft.y	≥	(createParams.winHeight – 1)/2
inBufDesc[RIGHT].bufPlanes[0].frameROI.topLeft.x	≥	(createParams.winWidth – 1)/2
inBufDesc[RIGHT].bufPlanes[0].frameROI.topLeft.y	≥	(createParams.winHeight – 1)/2

These constraints are actually similar to any constraints associated to a filtering type of function.

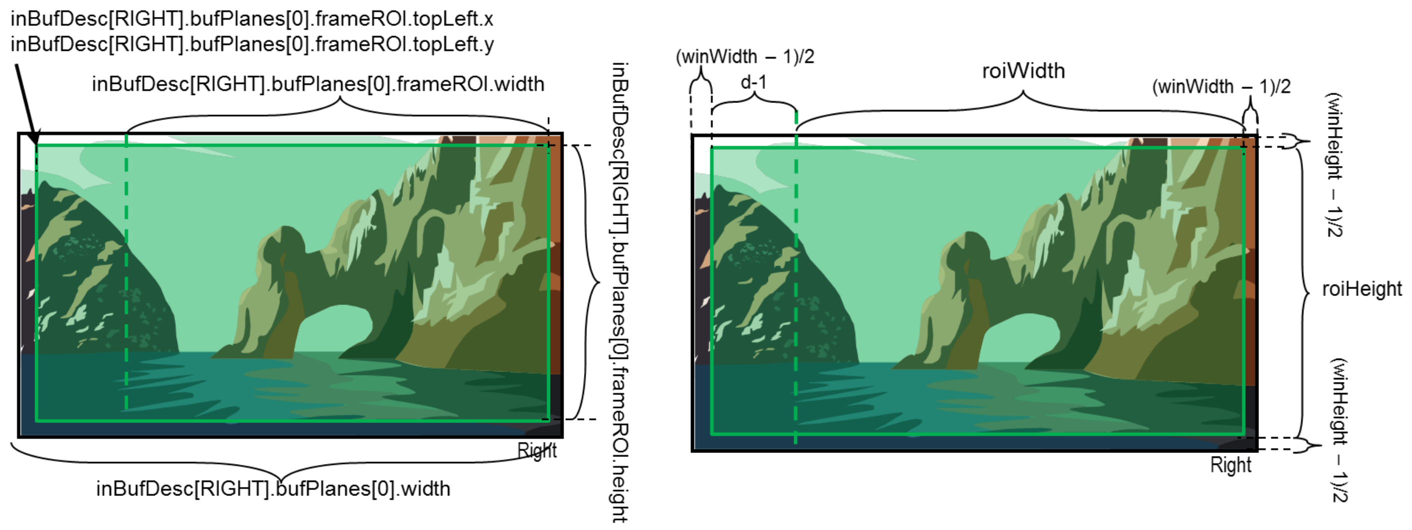
If `createParams.searchDir= DISPARITY_TI_LEFT_TO_RIGHT` is selected and if desired output disparity range is `[0, numDisparities -1 ]` then the constraints become:

inBufDesc[LEFT].bufPlanes[0].frameROI.topLeft.x	≥	(createParams.winWidth – 1)/2 + ***createParams.numDisparities – 1***
inBufDesc[LEFT].bufPlanes[0].frameROI.topLeft.y	≥	(createParams.winHeight – 1)/2
inBufDesc[RIGHT].bufPlanes[0].frameROI.topLeft.x	≥	(createParams.winWidth – 1)/2
inBufDesc[RIGHT].bufPlanes[0].frameROI.topLeft.y	≥	(createParams.winHeight – 1)/2

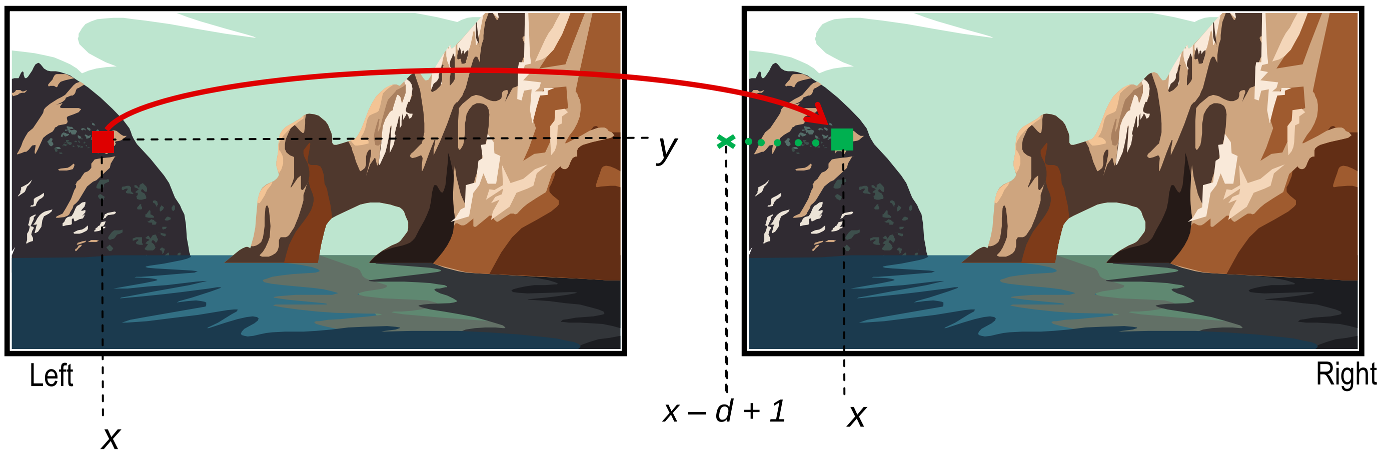
The figure below illustrates the constraints on the left image in case `createParams.searchDir= DISPARITY_TI_LEFT_TO_RIGHT`:


inBufDesc[LEFT].bufPlanes[0].frameROI.topLeft.x
inBufDesc[LEFT].bufPlanes[0].frameROI.topLeft.y
inBufDesc[LEFT].bufPlanes[0].frameROI.width
inBufDesc[LEFT].bufPlanes[0].frameROI.height
Left
inBufDesc[LEFT].bufPlanes[0].width
(winWidth -1)/2 + d - 1
roiWidth
(winWidth – 1)/2
roiHeight
(winHeight – 1)/2
Left
width

The figure below illustrates the constraints on the right image in case `createParams.searchDir= DISPARITY_TI_LEFT_TO_RIGHT`:


inBufDesc[RIGHT].bufPlanes[0].frameROI.topLeft.x
inBufDesc[RIGHT].bufPlanes[0].frameROI.topLeft.y
inBufDesc[RIGHT].bufPlanes[0].frameROI.width
inBufDesc[RIGHT].bufPlanes[0].frameROI.height
inBufDesc[RIGHT].bufPlanes[0].width
Right
(winWidth – 1)/2
d-1
roiWidth
(winWidth – 1)/2
(winHeight – 1)/2
roiHeight
(winHeight – 1)/2
Right

As you can observe, the main difference between the two cases `DISPARITY_TI_RIGHT_TO_LEFT` and `DISPARITY_TI_LEFT_TO_RIGHT` is that for the later, `topLeft.x` of the left image has an additional term of `createParams.numDisparities – 1`. This additional term is to guarantee valid disparities in the left side of the image. The extra term can be smaller than `createParams.numDisparities – 1` or even be 0, but then some disparities will be wrong within a band of `createParams.numDisparities – 1` pixels wide in the left-hand side of the image. The figure below depicts such situation where the left pixel coordinates (x,y) has its x coordinates strictly smaller than d – 1, resulting in comparison with pixel (x – d + 1, y) in the right image that falls outside of the frame.



If any of these constraints is not followed, the function will still execute but some outputs values along the border or within either the left or right hand side of the output will be wrong.

At creation time, the members `createParams.imageFrameWidth` and `createParams.imageFrameHeight` are passed through the creation params structure `DISPARITY_TI_CreateParams`. These values correspond to the actual ROI's width and height, depicted by the blue region in Figure 1, which will be processed by `algProccess()`. Using the values, the underlying creation function (initiated by algInit) will allocate the onchip memory accordingly and partition the frame in blocks. The dimensions of the blocks can be obtained after creation, by calling the `algControl()` function, which will return `controlOutParams->outputBlockWidth` and `controlOutParams->outputBlockHeight`. Note that for this function, the ROI's width and height can be changed after creation since the ROI dimensions are passed to `inBufDesc[0].bufPlanes[0].frameROI.width` and `inBufDesc[0].bufPlanes[0].frameROI.height`, which are passed to algProcess(). Normally for other applets, the `inBufDesc[]…frameROI.width` and `inBufDesc[]….frameROI.height` values must exactly match the values used in `createParams.imageFrameWidth` and `createParams.imageFrameHeight`. In other words, once the ROI width and height are set at create time, it is not possible to use different values at process time without causing errors. For this applet though, it is possible to re-specify these values by just setting `inBufDesc[0].bufPlanes[0].frameROI.width` and `inBufDesc[0].bufPlanes[0].frameROI.height` with new ones. The only constraint is that these new dimensions must be multiple of `controlOutParams->outputBlockWidth` and `controlOutParams->outputBlockHeight`, which are returned by `algControl()`.

Application to stereo-vision

This function, along with remap to rectify lens distortions, image misalignments and census transform, forms a complete stereovision algorithm. Example of such an application can be found in directory algorithms/stereoVision in the EVE SW release. A DSP post-processing stage to clean up the disparity map from invalid, noisy outputs is also available in the DSP APP release.

3.22.1 Data Structures

3.22.1.1 DISPARITY_TI_CreateParams

Description

This structure defines the creation parameters for disparity applet.

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`imageFrameWidth`	`uint16_t`	Input	Width in number of census codewords of the ROI image for which the disparity between left and right image will be produced.
`imageFrameHeight`	`uint16_t`	Input	Height in number of lines of the ROI image for which the disparity between left and right image will be produced.
`inputBitDepth`	`uint8_t`	Input	Number of bits in each left and right image's pixel or codewords (depending whether costMethod= `DISPARITY_TI_SAD` or `DISPARITY_TI_HAMMING_DIST`). Currently only 32-bits is supported.
`winWidth`	`uint8_t`	Input	Width of the support window in which costs are aggregated using either SAD or hamming distance.
`winHeight`	`uint8_t`	Input	Height of the support window in which costs are aggregated using either SAD or hamming distance.
`numDisparities`	`uint8_t`	Input	Number of disparities for which the costs are calculated. For each pixel, the disparity corresponding to the minimum cost is returned. When `searchDir= DISPARITY_TI_LEFT_TO_RIGHT`, disparity search is done by fixing a pixel in left image and then searching for the minimum cost pixel in the right image, along the same epilpolar line. Setting `searchDir= DISPARITY_TI_BOTH` will also do the search from" right image "to" left image.

Field	Data Type	Input/ Output	Description
`disparityStep`	`uint8_t`	Input	Disparity step allows to down-sample the number of disparities for which the cost is calculated, resulting in faster computation (but with some loss of quality).
`costMethod`	`uint8_t`	Input	Currently only `DISPARITY_TI_HAM_DIST` is supported.
`searchDir`	`uint8_t`	Input	`DISPARITY_TI_RIGHT_TO_LEFT` `DISPARITY_TI_LEFT_TO_RIGHT`
`outputCostType`	`uint8_t`	Input	`DISPARITY_TI_MINCOST:` output the minimum cost value associated to each pixel's disparity value. `DISPARITY_TI_MIN_ADJACENT_COSTS:` output the minimum cost value and the adjacent values associated to each pixel's disparity value. In total 3 cost planes are written out: the minimum cost plane, the previous cost plane and the next cost plane.

3.22.1.2 DISPARITY_TI_OutArgs

Description

This structure contains all the parameters which are produced as an output by the disparity applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules
`activeImageWidthLeft`	`uint16_t`	Output	Width in bytes of the area that will be accessed by the EDMA when reading the left image frame. When searchDir= `DISPARITY_TI_LEFT_TO_RIGHT` it is equal to `(imgFrameWidth + winWidth - 1)*inputBitDepth/8` bytes and when When searchDir= `DISPARITY_TI_RIGHT_TO_LEFT`, it is equal to `(imgFrameWidth + winWidth – 1 + numDisparities - 1)*inputBitDepth/8` bytes

Field	Data Type	Input/ Output	Description
`activeImageWidthRight`	`uint16_t`	Output	Width in bytes of the area that will be accessed by the EDMA when reading the right image frame. When searchDir= `DISPARITY_TI_LEFT_TO_RIGHT` it is equal to `(imgFrameWidth + winWidth – 1 + numDisparities - 1)*inputBitDepth/8` bytes and when When searchDir= `DISPARITY_TI_RIGHT_TO_LEFT`, it is equal to `(imgFrameWidth + winWidth – 1)*inputBitDepth/8` bytes
`activeImageHeight`	`uint16_t`	Output	Height in number of rows of the area that will be accessed by the EDMA when reading the right and left frame. For this function, it should always be equal to (imgFrameHeight + winHeight - 1)
`outputBlockWidth`	`uint16_t`	Output	The output frame will be written block-wise. Each block will be of dimension outputBlockWidth x outputBlockHeight. The value of outputBlockWidth is returned in this parameter and will be a divisor of imgFrameWidth.
`outputBlockHeight`	`uint16_t`	Output	The output frame will be written block-wise. Each block will be of dimension outputBlockWidth x outputBlockHeight. The value of outputBlockHeight is returned in this parameter and will be a divisor of imgFrameHeight.

3.22.1.3 DISPARITY_TI_ControlOutParams

Description

This structure contains all the parameters which are produced as an output by the disparity applet's algControl() function.

Fields

Field	Data Type	Input/ Output	Description
`algParams`	`IALG_Params`	Output	Commom structure for algParams ivision modules
`outputBlockWidth`	`uint16_t`	Output	The output frame will be written block-wise. Each block will be of dimension outputBlockWidth x outputBlockHeight. The value of outputBlockWidth is returned in this parameter and will be a divisor of imgFrameWidth.

Field	Data Type	Input/ Output	Description
`outputBlockHeight`	`uint16_t`	Output	The output frame will be written block-wise. Each block will be of dimension outputBlockWidth x outputBlockHeight. The value of outputBlockHeight is returned in this parameter and will be a divisor of imgFrameHeight.

3.22.2 Constraints

- createParams.imageFrameWidth must be multiple of 32.
- createParams.numDisparities must be multiple of 8.
- All constraints listed in the frame layout section.
- inputBitDepth must be 32.

3.22.3 References

- http://chrisjmccormick.wordpress.com/2014/01/10/stereo-vision-tutorial-part-i/
- http://www.vision.caltech.edu/bouguetj/calib_doc/

3.23 Feature Matching

This applet performs brute force search for matching features in one image/frame with features from another image/frame. The matching is performed based on Hamming distance measure. The applet is hence suitable for binary string based descriptors like ORB, BRIEF etc.

The input to the applet is two lists of feature descriptors – one for the first frame and one for the second frame. It takes the descriptor of one feature in first list and is matched with all other features in second list using Hamming distance as norm/metric for comparison. The index to one with minimum Hamming distance is reported as a match if the minimum Hamming distance is lower than user specified threshold and satisfies the ratio test for match confidence measure specified.

The quality of feature matching can be controlled by specifying a '`matchConfidence`' parameter. For each feature descriptor from first list, we find two best features matches from the second list – say with hamming distances of `minDist0` and `minDist1`. The descriptor from second list, which had the minimum hamming distance (`minDist0`), is declared as a confident match only if it satisfies the below ratio test criteria:

```
minDist0 <= (1 – matchConfidence) * minDist1
```

3.23.1 Data Structures

3.23.1.1 FEATURE_MATCHING_TI_CreateParams

Description

This structure defines the creation parameters for the Feature Matching applet

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`descriptorLength`	`Uint16_t`	Input	Length of each feature descriptor in bytes.

3.23.1.2 FEATURE_MATCHING_TI_InBufOrder

Description

User provides most of the infomration through buffer descriptor during process call.

The below enums define the purpose of the input buffer.

Fields

Enumeration	Description
`FEATURE_MATCHING_TI_BUFDESC_IN_FEATURE1`	This buffer descriptor should point to a buf descriptor pointing to list of feature descriptors for which correspondence needs to be found. The input should be a byte array (uint8_t data type).
`FEATURE_MATCHING_TI_BUFDESC_IN_FEATURE2`	This buffer descriptor should point to a buf descriptor pointing to list of feature descriptors from which correspondence needs to be searched for. The input should be a byte array (uint8_t data type).

Enumeration	Description
`FEATURE_MATCHING_TI_BUFDESC_IN_TOTAL`	This indicates total number of input buffers expected.

3.23.1.3 *FEATURE_MATCHING_TI_OutBufOrder*

Description

User provides most of the infomration through buffer descriptor during process call.

Below enums define the purpose of out buffer.

Fields

Enumeration	Description
`FEATURE_MATCHING_TI_BUFDESC_OUT_CORRESPONDENCE`	This buffer descriptor should point to a buffer capable of holding the list of indices from second feature descriptor list which correspond to each feature descriptor in the first list. The output indices are of uint16_t data type.
`FEATURE_MATCHING_TI_BUFDESC_OUT_TOTAL`	This indicates total number of output buffers expected.

3.23.1.4 *Control Command Indices*

Description

This following Control Command indices are supported for Feature matching applet.

Fields

Field	Description
`FEATURE_MATCHING_TI_CONTROL_GET_BUF_SIZE`	The algorithm expects the input and output buffers to be satisfy certain contraints. These details are communicated to the user using a control call with this index. Given the input control parameters in FEATURE_MATCHING_TI_CtrlParams, the control call return the buffer constraints through the output parameters of the same structure. This control call is optional in case user can statically provide bigger input and output buffers. In case input buffer can have extra space for upto 128 descriptors and space for extra 128 indices (16-bit) in output buffer, the control call need not be made. The data in the padded region of input buffers does not affect the valid outputs. The data produced in extra region in output buffer should be ignored.

3.23.1.5 *FEATURE_MATCHING_TI_CtrlParams*

Description

This structure contains all the parameters needed for control call of this applet.

Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	This is the input buffer which is provided in process call. Control function will determine the region for which this kernel would be running for the input dimensions givenby the user. User is expected to allocate this much memory for inout buffer
`numDescriptors1`	`uint16_t`	Input	Number of feature descriptors in feature descriptor list 1.
`numDescriptors2`	`uint16_t`	Input	Number of feature descriptors in feature descriptor list 2.
`in1BufSize`	`uint32_t`	Output	Minimum buffer size in bytes required for Feature 1 input buffer.
`in2BufSize`	`uint32_t`	Output	Minimum buffer size in bytes required for Feature 2 input buffer.
`outBufSize`	`uint32_t`	Output	Buffer size in bytes required for correspondence output.

3.23.1.6 FEATURE_MATCHING_TI_InArgs

Description

This structure contains all the parameters which are given as an input to Feature Matching applet during each process call.

Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_InArgs`	Input	Common inArgs for all ivison based modules
`minDistanceThres`	`uint16_t`	Input	Minimum distance between features for declaring a match.
`matchConfidence`	`uint16_t`	Input	Parameter to control ratio test used for eliminating matches that are ambiguous - i.e. the best match (minDist0) and next best match (minDist1) are very close to one another. A match is declared only if minDist0 <= (1 - matchConfidence)*minDist1. The parameter value can vary from 0 to 1 and should be provided in U1.15 format

3.23.1.7 FEATURE_MATCHING_TI_OutArgs

Description

This structure contains all the parameters which are given as an output by Feature Matching applet during each process call.

Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules

3.23.2 Constraints

- The feature descriptors sizes should be in multiple of bytes. In case user wants to use the applet for feature descriptor size that is not a clean multiple of 8 bits, user should pad extra 0 bits to each feature descriptor to the next byte boundary in both feature descriptor lists.

Maximum feature descriptor size (`descriptorLength`) supported is 512 bytes.

3.24 Hough Transform for Circles

This applet implements circle detection using Hough transform for circles. For a given list of radii of circles to be detected, the applet generates the Hough plane for each of the radius and reports the Hough space co-ordinates (center co-ordinates) and the Hough space score for points in Hough space that have score greater than a user specified threshold.

The applet takes two inputs buffer - an input image and corresponding edge map image. The edge map should to be gray-scale image of same dimension as the input image of data type uint8_t, with each pixel indicating whether the corresponding pixel in the input image is an edge (edge map value of 1) or not (edge map value 0).

Padding requirement in inputs: Based on the image dimensions, the applet expects additional padding in both width and height dimensions in both input image and the edge map. The exact amount of padding is obtained by making a control call.

Output format: The output is an array of detected circle information for each radius to be detected. This array is present as part of the output argument structure (`HOUGH_FOR_CIRCLES_TI_OutArgs`). The actual number of circles detected for each radius is provided as another array (`numCircles`) as part of the output argument structure of the process call.

While detecting circles for a particular radius 'r', the Hough space for that particular radius will have complete information only within the central region within a border of 'r' on each side. Hence the applet detects circles only within this region. While performing detection for a list of radii, detection is performed only within the complete region for the largest radius.

To achieve reasonable performance, each edge point votes into the Hough plane for a particular radii (r) only at a point that is at a distance r in the direction of the gradient at the point. The direction of voting ideally depends on the type of circle one is trying to detect. In case we have a bright circle in dark background, one needs to vote exactly in the direction of the gradient. In case the circle to be detected is a dark circle in bright background, voting should be done in a direction exactly opposite to the gradient. In the absence of information on type of circle to be detected, the applet votes in both directions so as to be able to detect both types of circles.

The applet provides an option to user to vote into a down-scaled Hough space. Apart from original Hough space, user can select to vote into a Hough space that's effectively downscaled by accumulating votes over either an 2x2 or 4x4 or an 8x8 grid. High down scale factors can give sharper peaks in Hough space and can help avoid the need to non-maximum suppression within the radius plane.

3.24.1 Data Structures

3.24.1.1 HOUGH_FOR_CIRCLES_TI_CircleType

Description

The voting scheme in the Hough for circles implementation depends on the type of circle being analysed - whether the circular object to be detected is 'bright' or 'dark'. By 'bright' we mean a bright circular object on a dark background. The enum lists a possible set of options user can specify the type of circular objects to be detected by the applet.

Enumerations

Enumeration	Description
`HOUGH_FOR_CIRCLES_TI_DARK`	Dark circular object on a bright background.
`HOUGH_FOR_CIRCLES_TI_BRIGHT`	Bright circular object on a dark background.

Enumeration	Description
`HOUGH_FOR_CIRCLES_TI_ALL`	Both bright and dark circular objects.

3.24.1.2 HOUGH_FOR_CIRCLES_TI_HoughSpaceScaling

Description

The amount of downsampling to be performed in the output Hough space during analysis. In general, increasing the amount of downsampling can lead to better chances of detection during thresholding but at the cost of accuracy in the location of center co-ordinates. With higher downscaling in Hough space one may not be required to perform non-maximum suppression within a radius plane.

Enumerations

Enumeration	Description
`HOUGH_FOR_CIRCLES_TI_NO_SCALING`	No down scaling in Hough space.
`HOUGH_FOR_CIRCLES_TI_SCALE_2x2`	Down-scaling the Hough space by accumulating hough votes over a 2x2 grid.
`HOUGH_FOR_CIRCLES_TI_SCALE_4x4`	Down-scaling the Hough space by accumulating hough votes over a 4x4 grid.
`HOUGH_FOR_CIRCLES_TI_SCALE_8x8`	Down-scaling the Hough space by accumulating hough votes over a 8x8 grid.

3.24.1.3 HOUGH_FOR_CIRCLES_TI_CreateParams

Description

This structure defines the creation parameters for the Hough for circles applet.

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`maxPercentageEdges`	`uint8_t`	Input	Maximum percentage of edge points that is expected per frame. In case user does not have this information apriori, set the value to zero. If the value is set to zero the applet will ask for worst case memory for it's internal memtab.
`circleType`	`uint8_t`	Input	This parameter specifies whether the applet needs to detect 'bright' or 'dark' circular object or both. Here 'bright' means bright circular object on a dark background. Refer to HOUGH_FOR_CIRCLES_TI_CircleType

Field	Data Type	Input/ Output	Description
`houghSpaceScaling`	`uint8_t`	Input	This parameter specifies the amount of down-scaling to beperformed in output hough space for analysis for every radius. Refer to HOUGH_FOR_CIRCLE_TI_HoughSpaceScaliing.
`maxNumRadius`	`uint8_t`	Input	Maximum number of radius user wants to work with. The maximum value supported is HOUGH_FOR_CIRCLES_TI_MAX_RADIUS .

3.24.1.4 *HOUGH_FOR_CIRCLES_TI_InBufOrder*

Description

User provides most of the infomration through buffer descriptor during process call.

The below enums define the purpose of the input buffer.

Fields

Enumeration	Description
`HOUGH_FOR_CIRCLES_TI_BUFDESC_IN_IMAGE`	This buffer descriptor (inBufs->bufDesc[HOUGH_FOR_CIRCLES_TI_BUFDESC_IN_IMAGE]) should point to a buffer descriptor containing the image in which circles need to be detected.
`HOUGH_FOR_CIRCLES_TI_BUFDESC_IN_EDGEMAP`	This buffer descriptor (inBufs->bufDesc[HOUGH_FOR_CIRCLES_TI_BUFDESC_IN_EDGEMAP]) should point to a buffer descriptor containing a 8 bit grayscale image indicating positions of the edge points in the image. This buffer should be of same dimensions as the input image and should have one-to-one correspondence between pexels of both input image and edge map image.
`HOUGH_FOR_CIRCLES_TI_BUFDESC_IN_TOTAL`	This indicates total number of input buffers expected.

3.24.1.5 *HOUGH_FOR_CIRCLES_TI_CircleProperties*

Description

This structure contains all the parameters needed for specifying properties of circles to be detected by this applet.

Fields

Field	Data Type	Input/ Output	Description
`Radius`	`uint8_t`	Input	Radius of the circle to be detected. The maximum value supported is HOUGH_FOR_CIRCLES_TI_MAX_RADIUS

Field	Data Type	Input/ Output	Description
`houghScoreThreshold`	`uint8_t`	Input	The threshold to be used in hough space to declare a circle for the given radius.

3.24.1.6 HOUGH_FOR_CIRCLES_TI_InArgs

Description

This structure contains all the parameters which are given as an output by Hough for circles applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_InArgs`	Input	Common inArgs for all ivison based modules
`numRadius`	`uint8_t`	Input	Number of different radii for which hough transform based detection need to be performed for the current frame.
`circleProps`	`HOUGH_FOR_CIRCLES_TI_CircleProperties`	Input	This parameter describes the properties to be used for circle detection for each radius.

3.24.1.7 HOUGH_FOR_CIRCLES_TI_OutArgs

Description

This structure contains all the parameters which are given as an output by Hough for circles applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules
`circleInfo`	`HOUGH_FOR_CIRCLES_TI_CircleInfo`	Output	Coordinates of the centers detected for all radius for current frame
`houghScore`	`uint8_t`	Output	Hough Score of the centers detected for all radius

Remark

It is important to note that when circle type is HOUGH_FOR_CIRCLES_TI_CIRCLE_BOTH then the actual number following arrays will contain double the number of radius requested with initial enteries being for the BRIGHT circle followed by DARK circles. For example if circleType is HOUGH_FOR_CIRCLES_TI_CIRCLE_BOTH and numRadius =3, numCircles array will contain 6 valid enteries with first 3 for BRIGHT circles and next 3 for DARK circles. circleInfo and houghScore array also follow the same behavior

3.24.1.8 HOUGH_FOR_CIRCLES_TI_ControlInArgs

Description

This structure contains all the parameters needed for control call of this applet. Control call is used to get the input buffer requirement by this applet. It is important that input buffer is allocated with the correct size returned by control call. The applet assumes that extra padded region for edgemap would be filled with zero by the user to avoid voting for those points by this applet.

Fields

Field	Data Type	Input/ Output	Description
`imageWidth`	`uint16_t`	Input	Input image buffer width
`imageHeight`	`uint16_t`	Input	Input image buffer height.
`maxRadius`	`uint8_t`	Input	maxRadius

Remark

It is important that the extra padded region for edgemap should be filled with zero by the user to avoid voting for those points by this applet

3.24.1.9 HOUGH_FOR_CIRCLES_TI_ControlOutArgs

Description

This structure contains all the parameters that are returned by control call of this applet. It is important that the extra padded region for edgemap should be filled with zero by the user to avoid voting for those points by this applet.

Fields

Field	Data Type	Input/ Output	Description
`effImageWidth`	`uint16_t`	Output	Effective image width needed by this applet.
`effImageHeight`	`uint16_t`	Output	Effective image height needed by this applet.

Remark

It is important that the extra padded region for edgemap should be filled with zero by the user to avoid voting for those points by this applet

3.24.2 Recommendations

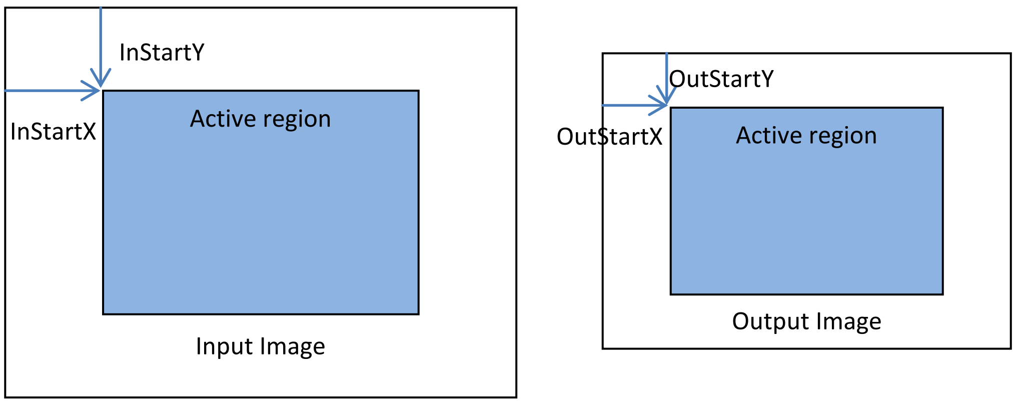
- Do not club analysis for very small radius with very large radius. The detection for small radii circles in border of the pcitures will be missed due to larger border region (with no detection) dictated by the largest radius in the list. Also performance for smaller radii will be poorer due to the presence of large radius.
- In case one does not want to decide the input buffer size dynamically by making the control call to get the buffer size, the user can statically allocate the input buffer with 48*(image width +1) bytes extra.
- In case the type of circle to be detected in a particular application is know apriori to be either a 'bright' circle or a 'dark' circle, it is recommended to set the circle type parameter accordingly. This will give faster execution of applet since voting needs to be done only to one half the points when circle type is known.
- The Hough score threshold to be set depends on the radius of the circle to be detected. For larger radius the threshold can be set higher.
- Non-maximum suppression is not performed on the output list that is provided. Suppression may not be required with a Hough plane for a particular radius for large down-scale settings. Suppression would still be required across the radius dimension.

3.24.3 Constraints

- Maximum number of different radii for which circle detection culd be performed in a single process call is limited to `HOUGH_FOR_CIRCLES_TI_MAX_NUM_RADIUS` (32).
- The radii of circles that could be reliable detected by this applet ranges between `HOUGH_FOR_CIRCLES_TI_MIN_RADIUS` (8) and `HOUGH_FOR_CIRCLES_TI_MAX_RADIUS` (40). These limits are not hard limit. Since the Hough space has a bit-depth of 8 bits, for circles with larger radius, the Hough score will be saturated to 255 and hence unique detection may or may not be possible based on the number of votes in the neighborhood.
- Maximum number of circles detected for each radius is limited to `HOUGH_FOR_CIRCLES_TI_MAX_NUM_CIRCLE_PER_RADIUS` (1000).
- The extra padded region for edgemap should be filled with zeros by the user to avoid voting for those points by this applet.

3.25 YUV Scalar

This applet accepts NV12 YUV (420 semi planner) as iput and re-sizes the it with user requested scale ratio. It supports both scale and scale down. Scale ratio shall be 0.25 =< Sclae ratio <= 2. User shall speficy the scale ratio in Q12 format.



3.25.1 Data Structures

3.25.1.1 YUV_SCALAR_TI_CreateParams

Description

This structure defines the creation parameters for YUV Scalar applet

Fields

Field	Data Type	Input/ Output	Description
visionParams	`IVISION_Params`	Input	Commom structure for vision modules
maxWidth	`uint16_t`	Input	Maximum image width supported
maxHeight	`uint16_t`	Input	Maximum image height supported
scaleRatioQ12	`uint16_t`	Input	re size / scale ratio in Q12 format
scalingMethod	`uint8_t`	input	Scaling methos to be used. Refer YUV_SCALAR_TI_ScalingMethods for suported methods
fracQFmt	`uint6_t`	Input	Q format for fraction pixel location represenatation (as well as filter co-efficients). Maximum supported values is 8
outStartX	`uint16_t`	Input	Horizontal offset in the ouput image (Refer the picture above)
outStartY	`uint16_t`	Input	Vertical offset in the ouput image (Refer the picture above)

3.25.1.2 YUV_SCALAR_TI_ScalingMethods

Description

This Enumaratior defines the scalling methods supported in YUV Scalar applet

Fields

Field	Description
YUV_SCALAR_TI_METHOD_BI_LINEAR_INTERPOLATION	Bi-Linear interpolation based scaling
YUV_SCALAR_TI_METHOD_MAX	Maximum Value for this enumarator

3.25.1.3 YUV_SCALAR_TI_ControlInParams

Description

This structure contains all the input parameters which controls the applet after creation. In the case of DiffOfGauss transform, it does not have any additional parameters than the default algParams from which it inherits the content.

Fields

Field	Data Type	Input/ Output	Description
algParams	`IALG_Params`	Input	Common params for all IALG based modules

3.25.1.4 YUV_SCALAR_TI_ControlOutParams

Description

This structure contains all the output parameters written by the control function the applet after creation. Mainly it contains output block dimensions which can be queried graph creation. The application should use these values to make sure that any ROI's width and height are multiple of the output block's width and height.

Fields

Field	Data Type	Input/ Output	Description
algParams	`IALG_Params`	Output	Common params for all IALG based modules
outputBlockWidth	`uint16_t`	Output	output block width
outputBlockHeight	`uint16_t`	Output	output block height

3.25.1.5 YUV_SCALAR_TI Control Commands

Field	Description
YUV_SCALAR_TI_GET_OUTPUT_BLOCK_DIM	Get output block's dimensions (width & height) derived during applet creation

3.26 Edge Detector

This applet implements a edge detector working at frame level. The routine accepts a gray scale input image and along with the threshold required for edge detection. The output dimension of this applet is lesser than input dimension by the 3x3. Output can be obtained in two different formats: one as a byte edge map whose each entry is either 0 (for non-edge pixel) or 255 (for edge-pixel) , the second format is a binary bit packed format. Whose bit if 1 indicates and edge and if 0 indicates a non-edge pixel.

3.26.1 Data Structures

3.26.1.1 EDGE_DETECTOR_TI_Method

Description

Edge detection can be done using different methods. Following is the enum for all the supported methods by this applet

Enumerations

Enumeration	Description
EDGE_DETECTOR_TI_METHOD_SOBEL	Use Sobel operator for edge Detection.
EDGE_DETECTOR_TI_METHOD_CANNY	Use Canny edge detection method for edge detection

3.26.1.2 EDGE_DETECTOR_TI_OutputFormat

Description

The format in which edge detection output is expected.

Enumerations

Enumeration	Description
EDGE_DETECTOR_TI_OUTPUT_FORMAT_BYTE_EDGEMAP	Output edge map image whose each pixel value is 255 if its an edge and 0 if its not an edge.
EDGE_DETECTOR_TI_OUTPUT_FORMAT_BINARY_PACK	Output edge map as a binary image with following mapping Byte0 -> Bit0 (LSB of the byte in binary image) Byte7 -> Bit7 (MSB of the byte in binary image) pixels :0 1 2 3 4 5 6 7 8 9 10 will be present as Binary : 7 6 5 4 3 2 1 0 15 14 23 12 11 10 9 8 .... and so on

3.26.1.3 EDGE_DETECTOR_TI_NormType

Description

Which norm to be used for edge detection.

Enumerations

Enumeration	Description

Enumeration	Description
`EDGE_DETECTOR_TI_NORM_L1`	Use L1 Norm ie.Mag =abs(X)+abs(Y)

3.26.1.4 EDGE_DETECTOR_TI_InBufOrder

Description

User provides most of the infomration through buffer descriptor during process call.

The below enums define the purpose of the input buffer.

Fields

Enumeration	Description
`EDGE_DETECTOR_TI_BUFDESC_IN_IMAGE`	This buffer descriptor (inBufs->bufDesc[EDGE_DETECTOR_TI_BUFDESC_IN_IMAGE]) should point to a buffer descriptor containing image for which edge detection needs to be performed.
`EDGE_DETECTOR_TI_BUFDESC_IN_TOTAL`	This indicates total number of input buffers expected.

3.26.1.5 EDGE_DETECTOR_TI_OutBufOrder

Description

User provides most of the infomration through buffer descriptor during process call. Below enums define the purpose of out buffer.

Fields

Enumeration	Description
`EDGE_DETECTOR_TI_BUFDESC_OUT_IMAGE_BUFFER`	EDGE_DETECTOR_TI_BUFDESC_OUT_IMAGE_BUFFER: This buffer will return edge detected image based in the format specified by the user using the enum. It is important to note that when method used is Canny the output is not the edges detectd. Instead output is an image with each pixel classified into three states: Pixel value 0 : For non edge pixels Pixel value 1 : For Pixels which are above low threshold and beloe the high threshold Pixel value 255 : For Pixels which are edge pixels
`EDGE_DETECTOR_TI_BUFDESC_OUT_TOTAL`	This indicates total number of output buffers expected.

3.26.1.6 EDGE_DETECTOR_TI_CreateParams

Description

This structure contains all the parameters needed at create time for edge detector applet

Fields

Field	Data Type	Input/ Output	Description
visionParams	`IVISION_Params`	Input	Commom structure for vision modules
method	`uint8_t`	Input	The method to be used for edge detection. Refer to EDGE_DETECTOR_TI_Method for supported methods
outputFormat	`uint8_t`	Input	The output format for the edge detector output. Refer to EDGE_DETECTOR_TI_OutputFormat for supported formats
normType	`uint8_t`	Input	Which type of norm to use to calculate magnitude. Refer to EDGE_DETECTOR_TI_NormType for supported norms

3.26.1.7 EDGE_DETECTOR_TI_InArgs

Description

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

Fields

Field	Description
iVisionInArgs	Common InArgs for all ivison based modules
threshold1	If sobel edge detector is used then this is the only threshold applied. But if Canny Edge detector is used then this is the first threshold for the hysteresis procedure.
threshold2	If sobel edge detector is used then this field is a dont care. But if Canny Edge detector is used then this is the second threshold for the hysteresis procedure.

3.27 Morphology

This applet performs grayscale and binary morphological filtering using a flat structuring element. The structuring element can be a generic mask or it can be a rectangular or cross mask. In case the user wishes to provide a generic structuring element, the element has to be provided as a mask of ones and zeros in an array of length se_width x se_height, where se_width and se_height are the dimensions of the structuring element. The supported morphological operations are dilation, erosion, opening, closing, top hat, bottom hat and morphological gradient.

In the case of grayscale morphology, the input and output images are assumed to be 8-bit grayscale images.

In the case of binary morphology, the input and output images are assumed to be inverted bit packed images. Each byte of the image will hold 8 pixel values. To detail, if Pi is the pixel of image at location i along the width (horizontal direction), then the image will ly in memory in the following format at ascending Bit locations:

P7 P6 ..... P0 P15 P14 .... P8 P23 P22 …. P16 P31 P30 …. P24 ……

3.27.1 Data Structures

3.27.1.1 MORPHOLOGY_TI_CreateParams

Description

This structure defines the creation parameters for the Morphology applet.

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules.
`srcType`	`uint8_t`	Input	Describes the bit depth for the input data. Refer MORPHOLOGY_TI_ImageType for valid enteries.

3.27.1.2 MORPHOLOGY_TI_InBufOrder

Description

User provides most of the information through buffer descriptor during process call. This enum defines the purpose of the input buffer.

Fields

Enumeration	Description
`MORPHOLOGY_TI_BUFDESC_IN_IMAGEBUFFER`	This buffer descriptor (inBufs->bufDesc[MORPHOLOGY_TI_BUFDESC_IN_IMAGEBUFFER]) should point to a buf descriptor pointing to input image data. The Input buffer should have worst case padding of 64 + 2*(se_height-1) + 1 rows. For Binary morphology, the width and pitch is expected to be in terms of number of bytes of the image. Therefore, if the image has a P pixel width, the width to be entered in the descriptor is P/8.
`MORPHOLOGY_TI_BUFDESC_IN_SEBUFFER`	This buffer descriptor (inBufs->bufDesc[MORPHOLOGY_TI_BUFDESC_IN_SEBUFFER]) should point to a buf descriptor pointing to the input structuring element only if the user is providing a generic mask.
`MORPHOLOGY_TI_BUFDESC_IN_TOTAL`	This indicates total number of input buffers expected.

3.27.1.3 MORPHOLOGY_TI_OutBufOrder

Description

User provides most of the information through buffer descriptor during process call. This enum defines the purpose of the output buffer.

Fields

Enumeration	Description
MORPHOLOGY_TI_BUFDESC_OUT_BUFFER	This buffer descriptor (outBufs->bufDesc[MORPHOLOGY_TI_BUFDESC_OUT_BUFFER]) should point to the output buffer. The output buffer should have worst case padding of 64 rows. For Grayscale morphology, the output buffer pitch should be a multiple of 64. For Binary morphology, the width and pitch is expected to be in terms of number of bytes of the image. Therefore, if the image has a P pixel width, the width to be entered in the descriptor is P/8.
MORPHOLOGY_TI_BUFDESC_OUT_TOTAL	This indicates total number of output buffers expected.

3.27.1.4 MORPHOLOGY_TI_ImageType

Description

This enum defines the type or bit depth of the input image.

Fields

Enumeration	Description
MORPHOLOGY_TI_BINARY_IMAGE	This denotes binary packed image.
MORPHOLOGY_TI_GRAYSCALE_IMAGE	This denotes grayscale 8-bit image.
MORPHOLOGY_TI_IMAGE_TYPES_MAX	This indicates total number of image types supported.

3.27.1.5 MORPHOLOGY_TI_StructuringElementShape

Description

This enum defines the type or shape of the structuring element.

Fields

Enumeration	Description
MORPHOLOGY_TI_CUSTOM_SE	This denotes generic mask.
MORPHOLOGY_TI_RECT_SE	This denotes rectangular mask.
MORPHOLOGY_TI_CROSS_SE	This denotes cross mask with the vertical strip lying at (se_width-1)/2 and the horizontal strip lying at (se_height-1)/2.
MORPHOLOGY_TI_SE_SHAPE_MAX	This indicates total number of masks supported.

3.27.1.6 MORPHOLOGY_TI_Operation

Description

This enum defines the morphological operation to be performed on the input image.

Fields

Enumeration	Description
`MORPHOLOGY_TI_DILATE`	This denotes dilation operation.
`MORPHOLOGY_TI_ERODE`	This denotes erosion operation.
`MORPHOLOGY_TI_OPEN`	This denotes open operation. It is Erosion followed by Dilation on the image.
`MORPHOLOGY_TI_CLOSE`	This denotes close operation. It is Dilation followed by Erosion on the image.
`MORPHOLOGY_TI_TOPHAT`	This denotes tophat operation. It is open(source) subtracted from the source.
`MORPHOLOGY_TI_BOTTOMHAT`	This denotes bottom hat operation. It is source subtracted from the close(source).
`MORPHOLOGY_TI_GRADIENT`	This denotes morphological gradient operation. It is erosion(source) subtracted from the dilation(source).
`MORPHOLOGY_TI_OPERATION_MAX`	This indicates total number of operations supported.

3.27.1.7 *MORPHOLOGY_TI_InArgs*

Description

This structure contains all the parameters which are given as an input to the Morphology applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_InArgs`	Input	Common inArgs for all ivison based modules
`morphologyOperation`	`uint8_t`	Input	Denotes the morphological operation. Refer MORPHOLOGY_TI_Operation for valid enteries.
`seShape`	`uint8_t`	Input	Denotes the structuring element shape. Refer MORPHOLOGY_TI_StructuringElementShape for valid enteries.
`seWidth`	`uint16_t`	Input	Width of the structuring element. For Grayscale morphology, maximum supported width is 13. For Binary morphology, only supported width is 3. In case of Cross mask, this value should be odd.

Field	Data Type	Input/ Output	Description
`seHeight`	`uint16_t`	Input	Height of the structuring element. For Grayscale morphology, maximum supported height is 13. For Binary morphology, only supported height is 3. In case of Cross mask, this value should be odd.

3.27.1.8 MORPHOLOGY_TI_OutArgs

Description

This structure contains all the parameters which are given as an output by Morphology applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules

3.27.2 Constraints

- Maximum width supported is 1920.
- For Grayscale Morphology, the maximum dimensions of the structuring element can be 13x13.
- For Binary Morphology, only a 3x3 structuring element is supported.
- For Cross Structuring Element, the SE width and SE height should be odd values.

3.28 Bin Image to list

This applet performs conversion of binary image to list. It also supports masking operation along with conversion of binary image to list. When masking operation is enabled user can provide a mask along with the binary image with 0 at locations which it doesn't want and 1 at other locations. In this case this applet will not take points for the list if mask for that point is set as zero.

3.28.1 Data Structures

3.28.1.1 BIN_IMAGE_TO_LIST_TI_ErrorType

Description

Errors returned by bin image to list applet.

Fields

Enumeration	Description
`BIN_IMAGE_TO_LIST_TI_ERRORTYPE_FORMAT_UNSUPPORTED`	Error returned when input data format given is not supported
`BIN_IMAGE_TO_LIST_TI_ERRORTYPE_QFORMAT_NOT_FEASIBLE`	Error returned if QFormat supplied by the user cant be applied
`BIN_IMAGE_TO_LIST_TI_ERRORTYPE_WIDTH_NON_MULTIPLE_OF_8`	Error returned when width is not multiple of 8
`BIN_IMAGE_TO_LIST_TI_ERRORTYPE_IMAGE_DIMENSION_UNSUPPORTED`	Error returned when image dimension given is not supported
`BIN_IMAGE_TO_LIST_TI_ERRORTYPE_IMAGE_DIMENSION_MISMATCH`	Error returned when image dimension mismatches

3.28.1.2 BIN_IMAGE_TO_LIST_TI_InputDataFormat

Description

Supported input data format by this applet.

Fields

Enumeration	Description
`BIN_IMAGE_TO_LIST_TI_INPUT_DATA_FORMAT_BIT_PACKED`	Input data is bit packed format

3.28.1.3 BIN_IMAGE_TO_LIST_TI_Order

Description

Enums to select the order of x and y in the output buffer.

Fields

Enumeration	Description

Enumeration	Description
`BIN_IMAGE_TO_LIST_TI_ORDER_XY`	Output will have X coordinate (upper 16 bit)followed by Y coordinate (lower 16 bits) for each edge point. Each X or Y is a 16 bit entry.
`BIN_IMAGE_TO_LIST_TI_ORDER_YX`	Output will have Y coordinate (upper 16 bit)followed by X coordinate (lower 16 bits) for each edge point. Each X or Y is a 16 bit entry

3.28.1.4 *BIN_IMAGE_TO_LIST_TI_InputMaskFormat*

Description

Enums to select the format of input mask image

Fields

Enumeration	Description
`BIN_IMAGE_TO_LIST_TI_INPUT_MASK_FORMAT_BYTE_MAP`	Input mask is a byte mask with 1 at all byte locations where user want to detect corners and 0 at other byte location

3.28.1.5 *BIN_IMAGE_TO_LIST_ListSuppressionMethod*

Description

Following enums should be used to select the method to be used when enableListSuppression in BIN_IMAGE_TO_LIST_TI_CreateParams is enabled

Fields

Enumeration	Description
`BIN_IMAGE_TO_LIST_TI_LIST_SUPPRESSION_BY_PERCENTAGE`	In this method the list suppression is based on the percentage of list elements user wants to pick from all the detected number of points. Percentage value should be provided via suppressionValue parameter present in BIN_IMAGE_TO_LIST_TI_InArgs. Value can range from 1 ot 100
`BIN_IMAGE_TO_LIST_TI_LIST_SUPPRESSION_BY_MAX_VALUE`	In this method the list suppression is based on the max value given by the user. Final number of list points detected will be less than the max value provided by the user. Max value should be provided via suppressionValue parameter present in BIN_IMAGE_TO_LIST_TI_InArgs.

3.28.1.6 *BIN_IMAGE_TO_LIST_TI_CreateParams*

Description

This structure defines the creation parameters for the this applet.

Fields

Field	Data Type	Input/ Output	Description

Field	Data Type	Input/ Output	Description
visionParams	IVISION_Params	Input	Commom structure for vision modules.
inputDataFormat	uint8_t	Input	Describes the format of the input data. It could be a byte binary image or could be bit packed binary image Refer BIN_IMAGE_TO_LIST_TI_InputDataFormat for valid supported enteries.
inputMaskFormat	uint8	Input	Describes the format of the mask to be applied on the input data. It could be a byte binary image or could be bit packed binary image Refer BIN_IMAGE_TO_LIST_TI_InputMaskFormat for valid supported enteries
enableMasking	uint8_t	Input	User can enable masking by separately providing a byte mask buffer with 1 at location where it wants to keep the points and 0 at other places
enableListSuppression	uint8_t	Input	User can enable suppression of the list by setting the value of this parameter to be 1. If list suppression is enable then based on user defined method itnernally we will uniformly downsample the image to reach the number of points or the ratio given by the user via BIN_IMAGE_TO_LIST_TI_InArgs
maxImageWidth	Uint16_t	Input	Maximum image width for which this applet will be used
maxImageHeight	Uint16_t	Input	Maximum image height for which this applet will be used

3.28.1.7 BIN_IMAGE_TO_LIST_TI_InBufOrder

Description

User provides most of the information through buffer descriptor during process call. This enum defines the purpose of the input buffer.

Fields

Enumeration	Description
BIN_IMAGE_TO_LIST_TI_BUFDESC_IN_BUFFER	This buffer descriptor (inBufs->bufDesc[BIN_IMAGE_TO_LIST_TI_BUFDESC_IN_BUFFER]) should point to a buf descriptor pointing to input binary image data which needs to be converted to list. This buffer format is decided by the inputFormat field in createParams. Width should be multiple of 8

Enumeration	Description
BIN_IMAGE_TO_LIST_TI_BUFDESC_IN_MASK_BUFFER	This buffer descriptor (inBufs->bufDesc[BIN_IMAGE_TO_LIST_TI_BUFDESC_IN_MASK_BUFFER]) should point to a buf descriptor pointing to the byte mask which needs to be applied to the image. Byte mask should contain 1 at location which you want to keep and 0 at other locations. This buffer is only required if you are enabling masking using enableMasking field of create params
BIN_IMAGE_TO_LIST_TI_BUFDESC_IN_TOTAL	This indicates total number of input buffers expected.

3.28.1.8 *BIN_IMAGE_TO_LIST_TI_OutBufOrder*

Description

User provides most of the information through buffer descriptor during process call. This enum defines the purpose of the output buffer.

Fields

Enumeration	Description
BIN_IMAGE_TO_LIST_TI_BUFDESC_OUT_LIST_XY	This buffer descriptor (outBufs->bufDesc[BIN_IMAGE_TO_LIST_TI_BUFDESC_OUT_LIST]) should point to the output buffer which will contain XY list after BIN_IMAGE_TO_LIST with thresholding. This list is expected to be in packed 32 bit format with order of X and Y determined by inArgs outputListOrder field. Size of this buffer should be the worst case size which will be equal to imageWidth * imageHeight * sizeof(uint32_t). If user given size of the buffer is less than the above mentioned then applet will return a warning saying not enough buffer. It is important to note that for the cases when buffer is not sufficient this applet behavior is undefined. Size of this buffer should be the worst cases size if suppresion is disabled. If suppression is enabled and method is BIN_IMAGE_TO_LIST_TI_LIST_SUPPRESSION_BY_MAX_VALUE then buffer should be of maxSize + 128 to take care of the extra buffer required by the applet
MORPHOLOGY_TI_BUFDESC_OUT_TOTAL	This indicates total number of output buffers expected.

3.28.1.9 BIN_IMAGE_TO_LIST_TI_InputDataFormat

Description

This enum defines the format supported for binary input data.

Fields

Enumeration	Description
`BIN_IMAGE_TO_LIST_TI_INPUT_DATA_FORMAT_BIT_PACKED`	Indicates Input Data is a bit masked data

3.28.1.10 BIN_IMAGE_TO_LIST_TI_InputMaskFormat

Description

This enum defines the format supported for mask to be used when enableMasking is enabled

Fields

Enumeration	Description
`BIN_IMAGE_TO_LIST_TI_INPUT_MASK_FORMAT_BYTE_MAP`	Indicates Input Data is a byte mask with 0 at location which needs to be masked from the list and 1 at other places

3.28.1.11 BIN_IMAGE_TO_LIST_TI_InArgs

Description

This structure contains all the parameters which are given as an input to the this applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionInArgs`	`IVISION_InArgs`	Input	Common inArgs for all ivison based modules
`outputListOrder`	`uint8_t`	Input	The order of X and Y in output List. Refer BIN_IMAGE_TO_LIST_TI_Order for valid vlaues
`outputListQFormat`	`uint8_t`	Input	The Qformat in which output is expected. The coordinates are shifted by outputListQFormat amount while packing the coordinates into list
`startX`	`uint16_t`	Input	X offset to be added to each X coordinate
`startY`	`uint16_t`	Input	Y offset to be added to each Y coordinate

Field	Data Type	Input/ Output	Description
`listSuppressionMethod`	`uint8_t`	Input	This parameter is only valid if enableListSuppression is enabled. User can choose two kind of suppression method as described in BIN_IMAGE_TO_LIST_ListSuppressionMeth od
`suppressionValue`	`uint32_t`	Input	This parameter is only valid if enableListSuppression is enabled. This parameter has different meaning based on the listSuppressionMethod selected. listSuppressionMethod = BIN_IMAGE_TO_LIST_TI_LIST_SUPPRES SION_BY_PERCENTAGE. This parameter should tell the percentage of points which should be picked from all the points detected. A value of 0 will return all the points as detected by the input binary image . listSuppressionMethod = BIN_IMAGE_TO_LIST_TI_LIST_SUPPRES SION_BY_MAX_VALUE. This parameter should be Maximum number of list points required. A value of 0 will return all the points as detected by the input binary image. Otherwise this value will be used to internally uniformly select points from all the points detected

3.28.1.12 BIN_IMAGE_TO_LIST_TI_OutArgs

Description

This structure contains all the parameters which are given as an output by this applet.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Pa rams`	Output	Commom structure for outArgs ivision modules
`numListPoints`	`uint32_t`	Output	Number of points in the the list

3.28.2 Constraints

- Image width should be multiple of 8.

3.29 Template Matching

This applet uses normalized cross correlation to find areas of an image that match (similar) to a template image.

Normalized cross correlation works as follow: given an input image `f` and a template image `t` of dimension `templateWidth x templateHeight`, for every location (u,v) in the input image, it calculates the following value `c(u,v)`

$$c(u,v) = \frac{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})(t(x-u,y-v) - \bar{t})}{\sqrt{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})^2 \sum_{x,y}(t(x-u,y-v) - \bar{t})^2}}$$

The index `x` and `y` iterates through the neighborhood patch of dimensions `templateWidth x templateHeight`, centered around the position `(u,v)`. $\overline{f_{u,v}}$ is the mean of the aforementioned neighborhood patch from the input image and $\bar{t}$ is the mean of the template image.

The position (u_{max},v_{max}) that coincides with the location where the template best matches the area of the input image corresponds to a local maximum for `c(u,v)`.

Note that the term $\sum_{x,y}(t(x-u,y-v) - \bar{t})^2$ is a constant that can be precomputed in advance or even be ignored as it doesn't affect the location of the local maximum of `c(u,v)`.
The template image `t` can also be normalized by subtracting its mean to produce a new template t'. Please refer to the Appendix C: Template Matching Normalization Function for more details.

Taking into consideration these simplifications, the equation becomes:

$$c(u,v) = \frac{\sum_{x,y}(f(x,y) - \bar{f}_{u,v}) \cdot t'(x-u,y-v)}{K \cdot \sqrt{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})^2}}$$

The constant K can be set to $\sqrt{\sum_{x,y}(t(x-u,y-v) - \bar{t})^2}$ or to 1 if the main goal is to find the local maximum.
Since EVE is a fixed-point processor, it does not directly compute `c(u,v)`. Indeed in order to avoid doing any division, it computes these two partial outputs:

$$num\_cc(u,v) = \sum_{x,y}(f(x,y) - \bar{f}_{u,v}) \cdot t'(x-u,y-v)$$

$$denom\_\mathrm{var}(u,v) = \sum_{x,y}(f(x,y) - \bar{f}_{u,v})^2$$

`num_cc(u,v)` is the numerator part of the cross-correlation `c(u,v)` and `denom_var(u,v)` is the the denominator part, which almost corresponds to the variance of the neighborhood patch. In order to increase the precision, the applets produce these two outputs as 32-bits number in Q format. The number of bits reserved for the fractional part is provided as a parameter `qShift` to the applet.

In order to obtain the final result, the following post-processing must be performed:

$$c(u,v) = \frac{num\_cc(u,v)}{K \cdot \sqrt{denom\_\mathrm{var}(u,v)}}$$

The reason why the applet does not perform this final step is that DSP can efficiently implement it thanks to its floating-point capability.

3.29.1 Data Structures

3.29.1.1 TEMPLATE_MATCHING_TI_CreateParams

Description

This structure defines the creation parameters for the Template Matching applet

Fields

Field	Data Type	Input/ Output	Description
`visionParams`	`IVISION_Params`	Input	Commom structure for vision modules
`imgFrameWidth`	`uint16_t`	Input	Width in pixels of the unsigned 8-bits input image
`imgFrameHeight`	`uint16_t`	Input	Height in number of lines of the unsigned 8-bits input image
`templateWidth`	`uint16_t`	Input	Width of the 16-bits 0-mean template in Q-format.
`templateHeight`	`uint16_t`	Input	Height of the 16-bits 0-mean template in Q-format.
`xDirJump`	`uint8_t`	Input	Jump in x direction while searching. Ignored at the moment. Always behave as if xDirJump= 1.
`yDirJump`	`uint8_t`	Input	Jump in y direction while searching. Ignored at the moment. Always behave as if yDirJump= 1.
`method`	`uint8_t`	Input	Template matching method. Currently only `TEMPLATE_MATCHING_TI_NCC` supported.
`qShift`	`uint8_t`	Input	Number of fractional bits of the Q-format used to format the output. qShift=*s* means Q(32-*s*).*s* format. For instance qShift=8, means Q24.8 format. Note that the greater qShift is, the more precise the outputs are but at the same time, it increases the risk of overflow during computation, leading to incorrect results. The following formula can be used to derive a safe value for qShift, for which overflow does not occur: qShift= min(7, (22 - log2(templateWidth*templateHeight))/2); A value greater than this recommended qShift value can still work, depending on the pixels values of both the template and input images. Please refer to appendix C for further details on how the formula is derived.

3.29.1.2 TEMPLATE_MATCHING_TI_InBufOrder

Description

User provides most of the infomration through buffer descriptor during process call.

The below enums define the purpose of the input buffer.

Fields

Enumeration	Description
`TEMPLATE_MATCHING_TI_BUFDESC_IN`	This buffer descriptor provides the actual image data required by algorithm. The data is composed of unsigned 8-bits data
`TEMPLATE_MATCHING_TI_BUFDESC_IN_TEMPLATE`	This buffer descriptor provides the template data required by algorithm. The data is composed of 16-bits data in Q format of a 0-mean template. A 0-mean template is obtained by subtrating the mean value from all the pixels. The number of fractional bits used for the Q format must match the parameter qShift passed to `TEMPLATE_MATCHING_TI_CreateParams`.
`TEMPLATE_MATCHING_TI_BUFDESC_IN_TOTAL`	This indicates total number of input buffers expected.

3.29.1.3 TEMPLATE_MATCHING_TI_OutBufOrder

Description

User provides most of the infomration through buffer descriptor during process call.

Below enums define the purpose of out buffer.

Fields

Enumeration	Description
`TEMPLATE_MATCHING_TI_BUFDESC_OUT`	This buffer is filled up by the algorithm and will contain two planes: - plane 0 contains the signed 32-bits numerator values of the cross-correlation. - plane 1 contains the unsigned 32-bits denominator values of the cross-correlation, corresponding to the variance of the neighborhood of the input location associated to the cross-correlation value.All the outputs value are in Q-format with 'qShift' factional bits, where 'qShift' is the parameter passed to TEMPLATE_MATCHING_TI_CreateParams.
`TEMPLATE_MATCHING_TI_BUFDESC_OUT_TOTAL`	This indicates total number of output buffers expected.

3.29.1.4 Control Command Indices

Description

This following Control Command indices are supported for Template matching applet.

Fields

Field	Description
`TEMPLATE_MATCHING_TI_CONTROL_GET_OUTPUT_BLOCK_DIM`	During applet creation time, the optimum outblock width and height are determined. It is possible to query these values after the applet is created by calling the control function with this command. The values are returned into the `TEMPLATE_MATCHING_TI_ControlOutParams` structure.

3.29.1.5 TEMPLATE_MATCHING_TI_ControlInParams

Description

This structure contains all the parameters needed for control call of this applet.

Fields

Field	Data Type	Input/ Output	Description
`algParams`	`IALG_Params`	Input	Common params for all IALG based modules.

3.29.1.6 TEMPLATE_MATCHING_TI_ControlOutParams

Description

This structure contains all the output parameters written by the control function the applet after creation. Mainly it contains output block dimensions which can be queried after graph creation. The application should use these values to make sure that any ROI's width and height are multiple of the output block's width and height.

Fields

Field	Data Type	Input/ Output	Description
`algParams`	`IALG_Params`	Output	Common params for all IALG based modules.
`outputBlockWidth`	`uint16_t`	Output	output block width
`outputBlockHeight`	`uint16_t`	Output	output block height

3.29.1.7 TEMPLATE_MATCHING_TI_OutArgs

Description

This structure contains all the parameters which are given as an output by TEMPLATE Matching applet during each process call.

Fields

Field	Data Type	Input/ Output	Description
`iVisionOutArgs`	`IVISION_Params`	Output	Commom structure for outArgs ivision modules

Field	Data Type	Input/ Output	Description
`activeImgWidth`	`uint16_t`	Output	activeImgWidth is primarily = imgFrameWidth + templateWidth - 1 and decided by applet to satisfy the internal DMA and kernel requirements. This is the actual number of horizontal pixels being accessed by EDMA. It is exported for user as informative
`activeImgHeight`	`uint16_t`	Output	activeImgHeight is primarily = imgFrameHeight + templateHeight - 1 and decided by applet to satisfy the internal DMA and kernel requirements. This is the actual number of vertical lines being accessed by EDMA. It is exported for user as informative
`outputBlockWidth`	`uint16_t`	Output	Output block width in number of pixels returned by BAM_createGraph(). That's useful information to understand performance.
`outputBlockheight`	`uint16_t`	Output	utput block height in number of rows returned by BAM_createGraph(). That's useful information to understand performance.

3.29.2 Constraints

- The template dimension templateWidth x templateHeight must be smaller than `TEMPLATE_MATCHING_TI_MAX_TEMPLATE_DIM`= 128*128= 16384 pixels.

3.30 CLAHE

This applet performs Contrast Limited Adaptive histogram equalization (CLAHE) on 8-bit input gray scale image to improve its contrast. It differs from ordinary histogram equalization in the respect that the adaptive method computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image. Contrast limiting prevents the overamplificaation of noise.

3.30.1 Data Structures

3.30.1.1 CLAHE_TI_ErrorType

Description

Error codes returned by the CLAHE algorithm.

Fields

Enumeration	Description
CLAHE_TI_ERRORTYPE_INVALID_IMAGE_DIMS	Image dimensions are beyond supported
CLAHE_TI_ERRORTYPE_INVALID_ROI_DIMS	ROI dimensions are beyond image dimensions
CLAHE_TI_ERRORTYPE_INVALID_ROI_ALIGN	ROI dimensions are not multiple of tile width or height
CLAHE_TI_ERRORTYPE_INVALID_TILE_DIMS	TILE dimensions are beyond supported value. Please try smaller tile
CLAHE_TI_ERRORTYPE_INVALID_TILE_ALIGN	TILE width and height are not aligned to 16
CLAHE_TI_ERRORTYPE_CLIP_LIMIT_BEYOND_RANGE	Clip limit value is beyond supported

3.30.1.2 CLAHE_TI_InBufOrder

Description

User provides most of the infomration through buffer descriptor during process call. Below enums define the purpose of buffer. There are 2 input buffers descriptors.

Fields

Enumeration	Description
CLAHE_TI_BUFDESC_IN_IMAGE_BUFFER	This buffer descriptor provides the actual image (Luma only) data required by algorithm.
CLAHE_TI_BUFDESC_IN_LUT_BUFFER	This buffer descriptor provides the LUT for each tile, that needs to be used for current frame processing.

3.30.1.3 CLAHE_TI_OutBufOrder

Description

User provides most of the infomration through buffer descriptor during process call. Below enums define the purpose of buffer. There are 2 output buffer descriptors.

Fields

Enumeration	Description
CLAHE_TI_BUFDESC_OUT_OBJ_BUFFER	This buffer descriptor has the propsecced image output
CLAHE_TI_BUFDESC_OUT_LUT_BUFFER	This buffer descriptor has the LUT for each tile that can be used for next frame

3.30.1.4 CLAHE_TI_CreateParams

Description

This structure contains all the parameters which CLAHE applet uses at create time

Fields

Field	Data Type	Input/ Output	Description
visionParams	`IVISION_Params`	Input	Common parmeters for all ivison based modules
maxWidth	uint16_t	Input	The maximum output width.
maxWidth	uint16_t	Input	The maximum output height.
tileWidth	uint16_t	Input	Tile Width for Histogram computation. This shall be multiple of 16 tileWidth*tileHeight s ahll be less than or equal to 16128
tileHeight	uint16_t	Input	Tile Hight for Histogram computation. This shall be multiple of 16 tileWidth*tileHeight s ahll be less than or equal to 16128

3.30.1.5 CLAHE_TI_InArgs

Description

This structure contains all the parameters which controls the applet at run time

Fields

Field	Data Type	Input/ Output	Description
iVisionInArgs	`IVISION_InArgs`	Input	Common inArgs for all ivison based modules
clipLimit	uint16_t	Input	Clip Limit for contrast adaptive contarst control.

3.30.1.6 CLAHE_TI_outArgs

Description

This structure contains all the parameters which are output from the applet

Fields

Field	Data Type	Input/ Output	Description
iVisionOutArgs	`IVISION_OutArgs`	Output	Common outArgs for all ivison based modules

4 Radar Applications

This section outlines the interface and feature details of different radar applications

4.1 FFT (Fast Fourier Transform)

This applet performs N point FFT where N can take following values 64, 128, 256 512 and 1024 points. Input to this applet is a list of signed complex number with real and imaginary data interleaved and stored in a 16 bit container. Output of the FFT can be 16-bits or 32-bits signed complex number with real and imaginary data interleaved.
Note that when output is 16-bits, a parameter can set the intermediary data to 32-bits for greater precision at the expense of slower execution time.
The output data is arranged in such a way so as to make next stage of radar processing which involves working across antenna easier. The output of this applet is described using a custom structure FFT_TI_BufferDescriptor. Apart from the FFT this applet also supports some radar specific pre and post processing of input and output data. The pre-processing includes sign extension, zero padding, interference zero out, dc-offset removal and windowing. Post processing includes Doppler correction.

4.1.1 Data Structures

4.1.1.1 FFT_TI_ErrorType

Description

Error codes returned by the FFT algorithm.

Fields

Enumeration	Description
FFT_TI_ERRORTYPE_UNSUPPORTED_FFT_DIMENSION	FFT is supported only for pre-defined number of points described in FFT_TI_NumPoints. Any other type will return error.
FFT_TI_ERRORTYPE_UNSUPPORTED_IRREGULARITY FREQUENCY	The freqency of irregularity should not be in such a way that jump from the base pointer to the next pointer where irregularity occurs is more than maxumum pitch supported by DMA
FFT_TI_ERRORTYPE_HORZ_FFT_DOPPLER_CORR_NOTSUPPORTED	For horizontal direction FFT doppler correction is not supported
FFT_TI_ERRORTYPE_HORZ_FFT_NUM_CHUNK_INVALID	Horizontal direction FFT only supports numChunk equal to 1.
FFT_TI_ERRORTYPE_HORZ_FFT_DOPPLER_CORR_NOTSUPPORTED	For horizontal direction FFT doppler correction is not supported
FFT_TI_ERRORTYPE_VERT_FFT_DCOFFSET_NOTSUPPORTED	For vertical direction FFT DC offset cacluation and application is not supported
FFT_TI_ERRORTYPE_VERT_FFT_SIGN_EXTENSION_NOTSUPPORTED	For vertical direction FFT sign extension is not supported
FFT_TI_ERRORTYPE_VERT_FFT_INTERFERENCE_ZEROOUT_NOTSUPPORTED	For vertical direction FFT interference zero out is not supported

Enumeration	Description
FFT_TI_ERRORTYPE_VERT_FFT_UNSUPPORTED_PITCH	ertical direction FFT only supports a maximum pitch of 32767 bytes
FFT_TI_ERRORTYPE_VERT_FFT_UNSUPPORTED_OFFSET_BW_ANTENNAS	Vertical direction FFT offset between anetnnas should be same as number of points in chunk times size of each element.
FFT_TI_ERRORTYPE_VERT_FFT_UNSUPPORTED_FFT_DIMENSION	Vertical direction FFT some of the higher dimension FFT is not supported
FFT_TI_ERRORTYPE_VERT_FFT_UNSUPPORTED_IRREGULARITYFREQUENCY	Vertical direction FFT should not have irregular offsetbetween antenna
FFT_TI_ERRORTYPE_VERT_FFT_INVALID_OUTPUT_PITCH	Unsupported output pitch for the vertical FFT
FFT_TI_ERRORTYPE_FFT_UNSUPPORTED_FFT_DIRECTION	Unsupported FFT direction
FFT_TI_ERRORTYPE_UNSUPPORTED_ZERO_PADDING	Indicates unsupported zero padding amount. Number of points for zero padding + number of actual points should be power of 2 and should lie between FFT_TI_NUM_POINTS_64 and FFT_TI_NUM_POINTS_1024. Also number of points for zero padding should be multiple of VCOP_SIMD_WIDTH(8)
FFT_TI_ERRORTYPE_INVALID_DATABITS	Unsupported FFT_TI_CreateParams->inDataValidBits field

4.1.1.2 FFT_TI_InBufOrder

Description

User provides most of the information through buffer descriptor during process call. Below enums define the purpose of buffer. There are maximum 3 input buffers descriptors.

Fields

Enumeration	Description
FFT_TI_BUFDESC_IN_LISTBUFFER	This buffer descriptor provides the pointer to a buffer whose FFT needs to be calculated. This buffer is described using the structure define below. Kindly refer FFT_TI_BufferDescriptor to It is important to note that apart from the only following enteries from this structure are used by this applet : bufDesc[]->bufPlanes[].buf bufDesc[]->bufPlanes[].accessMask
FFT_TI_BUFDESC_IN_WINDOWING_COEFF_BUF	This buffer descriptor provides the pointer to a buffer, which holds the windowing coefficients used at windowing stage. The windowing coefficients are 16-bit signed real numbers with the size of array being equal to the dimension in the direction of FFT. This buffer is only required if windowing is enable by setting FFT_TI_InArgs->enableWindowing to 1. This buffer is of dimension [RANGE_DIM] when FFT direction is horizontal and of dimension [DOPPLER_DIM] when FFT direction is vertical

Enumeration	Description
FFT_TI_BUFDESC_IN_DOPPLER_CORRECTION_BUF	This buffer descriptor provides the pointer to a buffer which holds doppler correction array. Each entry of this buffer is a complex number with 16-bit signed real and imaginary parts interleaved. This buffer is only required if doppler correction is enable by setting FFT_TI_InArgs->enableDopplerCorrection to 1. Doppler correction is only done when FFT direction is vertical. Dimension of this buffer should be [NUM_ANTENNAS][DOPPLER_DIM]

4.1.1.3 *FFT_TI_OutBufOrder*

Description

User provides most of the information through buffer descriptor during process call. Below enums, define the purpose of out buffer.

Fields

Enumeration	Description
FFT_TI_BUFDESC_OUT_BUFFER	This buffer descriptor provides the output buffer to store the data after FFT processing. This buffer is described using the structure define below. Kindly refer FFT_TI_BufferDescriptor in outArgs to know the description of this buffer. It is important to note that apart from the only following enteries from this structure are used by this applet : bufDesc[]->bufPlanes[].buf bufDesc[]->bufPlanes[].accessMask bufDesc[]->bufPlanes[].width : This field indicates the pitch in bytes of the output buffer. This field is used only when FFT direction is FFT_TI_DIRECTION_VERTICAL. if the value for this is set to zero then the pitch will be calculated internally. For this case size of this buffer should be same as size of input buffer. If user wants to control the pitch of output buffer then user should first call the Control call to figure out the minimum pitch required and user is expected to give pitch which is greater or equal to the minimum pitch. The maximum supported pitch is 32767 bytes.

4.1.1.4 *FFT_TI_ControlCommand*

Description

Control command can be used to get/set some of the parameters for this applet. User should call the ivision->algControl call with following command ID. This is an optional call. Below enums define the control commands supported for this applet.

Fields

Enumeration	Description

Enumeration	Description
FFT_TI_CONTROL_GET_OUTPUT_BUFDESCRIPTOR	This control command can be used to get the dimension of the output buffer. User should provide FFT_TI_InArgs as inParams and FFT_TI_BufferDescriptor as outParams while using this control command using ivision->algControl command API. This control command should be called to know the memory required for output buffer along with the pitch for the output buffer

4.1.1.5 *FFT_TI_Direction*

Description

The directions in which FFT can be calculated. Below enums define for this.

Fields

Enumeration	Description
FFT_TI_DIRECTION_HORIZONTAL	FFT is calculated along the direction of 1st dimension (horizontal) of buffer
FFT_TI_DIRECTION_VERTICAL	FFT is calculated along the direction of 2nd dimension (vertical) of buffer

4.1.1.6 *FFT_TI_NumPoints*

Description

The number of point in the direction of FFT. Below enums define for this.

Fields

Enumeration	Description
FFT_TI_NUM_POINTS_1024	Number of point of along the direction of FFT is 1024
FFT_TI_NUM_POINTS_512	Number of point of along the direction of FFT is 512
FFT_TI_NUM_POINTS_256	Number of point of along the direction of FFT is 256
FFT_TI_NUM_POINTS_128	Number of point of along the direction of FFT is 128
FFT_TI_NUM_POINTS_64	Number of point of along the direction of FFT is 64

4.1.1.7 *FFT_TI_ContainerFormat*

Description

Format of the input or output datatype. Below enums define for this

Fields

Enumeration	Description
FFT_TI_CONTAINER_FORMAT_16BIT	Each point of FFT (real and imaginary) is stored in 16 bit container. This is the only option for input data-type.
FFT_TI_CONTAINER_FORMAT_32BIT	Each point of FFT (real and imaginary) is stored in 32 bit container. Only applicable for output data-type.

4.1.1.8 *FFT_TI_BufferDescriptor*

Description

This structure is used to describe the buffers (input/output) to this applet. This descriptor represents the buffer in the unit of Antenna. We assume the same format across the antenna. Kindly refer section Appendix D: Radar Processing Data Flow to see some of the examples to understand the usage of this structure

Fields

Field	Data Type	Input/ Output	Description
numChunks	uint8_t	Input	If data for a single antenna cannot be represented within a maximum supported pitch then we will have to split the buffer into smaller chunk such that each chunk can represent the data of single antenna within with a pitch. This field represents this number of such chunks. For a regular FFT case this field should be set to 1. Note : It is important to note that when the direction of FFT is FFT_TI_DIRECTION_VERTICAL then the maximum supported pitch is 32767 bytes. Also when direction of FFT is horizontal numChunk should be set to 1
numHorzPoints	uint16_t	Input	Number of contigous points in Horizontal direction corresponding to a single antenna. This is an array of size equal to numChunks field in this structure. Horizontal direction FFT will be calculated along this dimension. For regulator FFT case this will correspond to the horizontal dimension of FFT
numVertPoints	uint16_t	Input	Number of contigous points in vertical direction corresponding to a single antenna. The contiguity is defined in terms of the pitch field in this structure. Vertical directionFFT will be calculated along this dimension. For regulator FFT case this will correspond to the vertical dimension of FFT
offsetBwAntennas	uint32_t	Input	Offset in terms of number of bytes to jump to next antenna data from the current base pointer. This field is also an array having numChunks enteries. This should be word aligned

Field	Data Type	Input/ Output	Description
offsetBwAntennas1	uint32_t	Input	This field should be used only when there is an irregularity in the offsetBwAntennas. This offset is in terms of number of bytes to jump to next antenna data from the current antenna data pointer base at the point when there is irregularity. This field is only used if irregularityFrequency != 0. This field is also an array having numChunks enteries. This should be word aligned
irregularityFrequency	uint16_t	Input	This field is to indicate after how many antenna there is a irregularity in offsetBwAntennas. A value of zero will assume a regular case where all the offset between antennas are the same. For example if the data is arranged as shown bellow A0xxxxA1xxxxA2xxxxyyyy A3xxxxA4xxxxA5xxxxyyyy Here Ak represents antenna data for a particular antenna and it will have range number of samples in it. For the above example following would be the values of above parameters : irregularityFrequency = 2, offsetBwAntennas = xxxx offsetBwAntennas1 = A2 + xxxx + yyyy
pitch	uint32_t	Input	Offset in terms of number of bytes to jump to the next doppler index corresponding to the same antenna.. This field is also an array having numChunks enteries. Pitch should be word aligned
numAntennas	uint16_t	Input	This field is meant for radar processing usecase and should be set to one for normal FFT calculation. This tells the total number of anetnna data which is coming with the buffer.

4.1.1.9 *FFT_TI_CreateParams*

Description

This structure contains all the parameters needed at create time for this applet

Fields

Field	Data Type	Input/ Output	Description
visionParams	IVISION_Params	Input	Common parmeters for all ivison based modules
fftDirection	uint8_t	Input	The direction in which FFT should be calculated. For supported values refer FFT_TI_Direction enum
inputContainerFormat	uint8_t	Input	This field is to indicate the container format of input data type. Currently only FFT_TI_CONTAINER_FORMAT_16BIT is supported.

Field	Data Type	Input/ Output	Description
outputContainerFormat	uint8_t	Input	This field is to indicate the container format of input data type. For supported values refer FFT_TI_ContainerFormat enum

4.1.1.10 *FFT_TI_InArgs*

Description

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

Fields

Field	Data Type	Input/ Output	Description
iVisionInArgs	iVisionInArgs	Input	Common InArgs for all ivision based modules
bufDescription	FFT_TI_Buffer Descriptor	Input	The input buffers to this applet comes as part of input buf descriptors, but the arrange of data in these buffer is described using FFT_TI_BufferDescriptor structure. This field hold this structure which describes the input buffers
enable32bitsIntermResults	uint8_t	Input	Set it to 1 to enable mode in which intermediary results are computed using 32-bits precision instead of 16-bits precision. Final outputs remain in 16-bits if outputContainerFormat was set to FFT_TI_CONTAINER_FORMAT_16BIT at creation time. Scaling is only applied in the last stage when results are written in 16-bits. Any scaling factor specified at other stages is automatically moved to the last stage by the applet. Note that this option is only applicable when outputContainerFormat is FFT_TI_CONTAINER_FORMAT_16BIT. When outputContainerFormat is FFT_TI_CONTAINER_FORMAT_32BIT, intermediary results are always computed in 32 bits and output is written out as 32-bits values with no scaling applied.
enableDcOffset	uint8_t	Input	Set it to 1 to enable DC offset calculation else set it to zero. DC offset will find the average in both real and imaginary direction and substract it from each input sample
enableInterferenceZeroOut	uint8_t	Input	Set it to 1 to enable interference zero out else set it to zero. This will make any input data point (both real and imaginary) greater than threshold specified by interferenceZeroOutThreshold to zero

Field	Data Type	Input/ Output	Description
enableWindowing	uint8_t	Input	Set it to 1 to enable Windowing else set it to zero. Windowing operation will multiply each point by the windowing coefficients before the FFT
enableDopplerCorrection	uint8_t	Input	Set it to 1 to enable doppler correction else set it to zero. Dopper correction will do a point to point complex multiplication of each sample.
enableOverFlowDetection	uint8_t	Input	Set it to 1 to enable overflow detection else set it to zero. If enabled the applet will return the scale factors to be applied at each stage as part of outArgs. if scale factors are zero for each stage then there is no overflow detected
interferenceZeroOutThreshold	uint16_t	Input	Threshold to be used for interference zero out. This is the maximum absolute value of the input data above which that sample would be set to zero. This is only required if enableInterferenceZeroOut is set to 1.
windowingScaleFactor	uint16_t	Input	Scale factor to be used at windowing operation. This factor is internally used to right shift (along with rounding) the output after multiplication by windowing coefficient. Mathematically if x is the output and n is the scaling factor then the output would be (x + 1 << (n - 1)) >> n. This is only required if windowing is enabled by setting enableWindowing = 1
dopplerCorrectionScaleFactor	uint16_t	Input	Scale factor to be used while doing doppler correction. This factor is internally used to right shift (along with rounding) the output after multiplication by doppler correction coefficients. Mathematically if x is the output and n is the scaling factor then the output would be (x + 1 << (n - 1)) >> n. This is only required if windowing is enabled by setting enableDopplerCorrection = 1
numPointsZeroPadding	uint16_t	Input	Number of points to be added to FFT for zero padding. numFFTPoints + numPointsZeroPadding should be power of 2 and should lie between FFT_TI_NUM_POINTS_64 and FFT_TI_NUM_POINTS_1024. Also numPointsZeroPadding should be multiple of VCOP_SIMD_WIDTH (8).
numValidBits	uint8_t	Input	Number of valid bits in input data. If numValidBits is 16 then sign extension is disabled
inDataRange	uint8_t	Input	This field is to indicate range of input data within the container format specified by containerFormat field. This information will be used to determine the stages in which overflow detection should be applied. This field is only required if enableOverFlowDetection is enabled (set to 1).

Field	Data Type	Input/ Output	Description
reserved	uint32_t	Input	This parameter is reserved parameters and is for future use.
scaleFactors	uint16_t	Input	Scale Factor to be used for each stage of FFT. This factor is internally used to right shift (along with rounding) the output after multiplication by twiddle factor (in Q15). Mathematically if x is the output and n is the scaling factor then the output would be (x + 1 << (n + 15 - 1)) >> (n + 15). Each stage of FFT grows by certain amount (1 bit or 2 bit for radix 2 or radix-4 respectively). Hence to avoid overflow each stage scale factor should be chosen such that input data is within 15 bit. It is important to note that in our implementation twiddle factors are stored in Q15 format and hence if a particular stage scale factos is n then its effective scaling applied at output would be (15 + n). Note that if enable32bitsIntermResults=1, in order to maintain the maximum dynamic range of 32-bits for the intermediary stages, no scaling is performed at these intermediate stages. Only when output is 16-bits, to prevent overflow in the last stage, scaling is applied. Note that the user can still specify scale factors for the intermediary stages. The applet will simply sum them up, add them to the last stage's scale factor.

4.1.1.11 FFT_TI_OutArgs

Description

This structure contains all the parameters which are given as an output by FFT applet at frame level

Fields

Field	Data Type	Input/ Output	Description
iVisionOutArgs	IVISION_OutArgs	output	Common outArgs for all ivison based modules
bufDescription	FFT_TI_BufferDescriptor	output	The output buffers to this applet comes as part of output buf descriptors, but the arrange of data in these buffer is described using FFT_TI_BufferDescriptor structure. This field hold this structure which describes the output buffer
scaleFactors	uint16_t	output	This array returns the scale factors to be applied at each stage when overflow detection is enabled. If scaleFactors for each stage is zero then no overflow has occured

4.2 Beam Forming

The purpose of this applet is to identify angular position of pre-detected object having range and velocity information. It expects a list of range and Doppler co-ordinates for the detections along with corresponding radar cube data having performed range and Doppler FFT. In order to find the angle, it also expects user to provide a steering vector corresponding to the angle. For example, if we have a radar cube of dimension 8(antennas) x 512(range) x 256 (Doppler) and want to identify the angle of a detected position in within -45 to +45 degree with a step of 5 degree. This would require 19 steering vectors because of 19 possible angles each of length 8 (antennas), also referred as steering matrix of dimension 19 x 8.
This applet computes energy at each prescribed angle by using the steering matrix, computes the angle having highest energy, and provides it to user as the angular position of the detection. The applet accepts list of detections in a buffer defined by *BEAM_FORMING_TI_Coordinate.* This same buffer format is used for input and output hence it has a placeholder for angle as well along with range and velocity information expected to be populated from user.

4.2.1 Data Structures

4.2.1.1 *BEAM_FORMING_TI_ErrorType*

Description

Error codes returned by the Beam forming applet algorithm.

Fields

Enumeration	Description
BEAM_FORMING_TI_ERRORTYPE_UNSUPPORTED_CONFIGURATION	Indicates an un-supported configuration
BEAM_FORMING_TI_ERRORTYPE_UNSUPPORTED_NUM_DETECTIONS	Indicates that numDetection is above max number of detections
BEAM_FORMING_TI_ERRORTYPE_UNSUPPORTED_STEERINGMATRIX_SIZE	Indicates that steering matrix size is more than supported
BEAM_FORMING_TI_ERRORTYPE_UNSUPPORTED_COORDINATE_FORMAT	Indicates that the coordinate buffer pointer is not supported for current configuration

4.2.1.2 *BEAM_FORMING_TI_CoordinateBufFormat*

Description

This enum is to select the type of format in which coordinate buffer(BEAM_FORMING_TI_BUFDESC_IN_COORDINATE_BUF) is given to this applet.

Fields

Enumeration	Description
BEAM_FORMING_TI_COORDINATE_BUF_FORMAT_1	In this format the pointer to coordinate buffer points to the coordinates of each detection which is stored as a list of coordinates structure defined by BEAM_FORMING_TI_Coordinates Angle field of this structure is optional

Enumeration	Description
BEAM_FORMING_TI_COORDINATE_BUF_FORMAT_2	In this format the pointer to coordinate buffer points to a buffer which holds the range and doppler coordinate in interleave manner. Where each range and doppler is 16 bit signed number and stored such that range lies in the upper 16 bit and doppler coordinate types in lower 16 bit.

4.2.1.3 *BEAM_FORMING_TI_Coordinates*

Description

This structure defines coordinates for a detection after the beam forming.

Fields

Field	Data Type	Input/ Output	Description
Velocity	uint16_t	Input	velocity for a particular detection.
Range	uint16_t	Input	Range dimension for a particular detection
Energy	uint16_t	Input	Energy calculated for a particular detection.
angleBin	uint16_t	Input	Bin number from the steering matrix corresponding to a particular detection. User is expected to do the mapping of bin to angle. This field is not required to be populated for this applet

4.2.1.4 *BEAM_FORMING_TI_InBufOrder*

Description

User provides most of the information through buffer descriptor during process call. Below enums define the purpose of buffer. There are 3 input buffers descriptors. For optimal performance, all the buffers should be aligned to 128 bytes boundary.

Fields

Enumeration	Description

Enumeration	Description

BEAM_FORMING_TI_BUFDESC_IN_ANTENNA_DATA_BUF	This buffer descriptor provides the pointer to a buffer which contains the antenna data. The data in this buffer is expected to be arranged as numAntennas x numDetections. Buffer is expected to look as shown below :
	`__|A0| A1| A2|........ |Ak |`
	`D0|`
	`D1|`
	`D2|`
	`|`
	`|`
	`Dn |`
	
	Here each entry is stored as 16-bit real and 16-bit imaginary part interleaved together. It is important to note that only following parameters of this structure are used by this applet :
	bufDesc[]->bufPlanes[].buf
	bufDesc[]->bufPlanes[].accessMask

BEAM_FORMING_TI_BUFDESC_IN_COORDINATE_BUF	This buffer descriptor is to provide the pointer to the coordinate buffer, which can be given in various different formats. Kindly refer BEAM_FORMING_TI_CoordinateBufFormat for various format supported by this applet
BEAM_FORMING_TI_BUFDESC_IN_STEERINGMATRIX_BUF	This buffer descriptor provides the pointer to a buffer holding steering matrix for beam forming. Steering matrix dimension is expected to be numAntennas x numAngles. Here each entry is stored as 16-bit real and 16-bit imaginary part interleaved together. Each row of steering matrix ccorresponds to one bin/angle. The row number is treated as ID for the angle, and assigned to BEAM_FORMING_TI_CreateParams->angleBin field in output buffer. User is expected to do the mapping of angleBin to actual angle. The steering matrix is expected to be arranged as follows Steering matrix bin_0 Steering matrix bin_1 : : Steering matrix bin_n Where offset between steering matrix of different bin is given by numAngles x numAntennas * sizeof(int16_t) * 2. As an example if user wants to detect from -20 to +20 degrees in azimuth and -10 to +10 degrees in elevation he should give a steering matrix having 41 x 21 rows where each row corresponds to the steering vector of a particular angle in azimuth or elevation. Final output will give the row via BEAM_FORMING_TI_Coordinates.angleBin field. User can then map this angle bin to the particular angle in azimuth or elevation

4.2.1.5 BEAM_FORMING_TI_OutBufOrder

Description

User provides most of the information through buffer descriptor during process call. Below enums define the purpose of out buffer. For optimal performance all the buffers should be aligned to 128 byte boundary.

Fields

Enumeration	Description
BEAM_FORMING_TI_BUFDESC_OUT_BUFFER	This buffer descriptor provides the output buffer to store the data after Beam Forming. This buffer contains the list of structure as defined in BEAM_FORMING_TI_Coordinates for each detection. It is important to note that only following entries from this structure are used by this applet : bufDesc[]->bufPlanes[].buf bufDesc[]->bufPlanes[].accessMask

4.2.1.6 BEAM_FORMING_TI_CreateParams

Description

This structure contains all the parameters which Beam Forming applet uses at create time

Fields

Field	Data Type	Input/ Output	Description
visionParams	`IVISION_Params`	Input	Common parameters for all ivison based modules
maxNumAntenna	uint16_t	Input	Maximum number of antennas in the system.
maxNumDetection	uint16_t	Input	Maximum number of detections. It is important to note that maximum number of detection supported is given by BEAM_FORMING_TI_MAX_NUM_DETECTIONS macro
maxNumAngle	uint16_t	Input	Maximum number of angles
coordinateBufFormat	uint8_t	Input	This field is used to indicate the format of the coordinate buffer. Kindly refer BEAM_FORMING_TI_CoordinateBufFormat enum for various supported formats

4.2.1.7 BEAM_FORMING_TI_InArgs

Description

This structure contains all the parameters which controls the applet at run time

Fields

Field	Data Type	Input/ Output	Description
iVisionInArgs	`IVISION_InArgs`	Input	Common inArgs for all ivison based modules

Field	Data Type	Input/ Output	Description
numDetections	uint16_t	Input	Number of detections
numAntennas	uint16_t	Input	Number of antennas
numAngles	uint16_t	Input	Number of angles/bin for beam forming
beamFormingScaling	uint16_t	Input	Scaling (right shift with rounding) that needs to be applied after doing complex matrix multiplication for beam forming. Mathematically if after complex multiplication either real/imaginary part is x then final output would be equal to (x + 1 << (n - 1)) >> n, where n is the scaling factor
energyScaling	uint16_t	Input	Scaling (right shift with rounding) that needs to be applied after calculating the energy in 32 bit to convert it to 16 bit. Mathematically if after energy computation energy is x then final output would be equal to (x + 1 << (n - 1)) >> n, where n is the scaling factor

4.2.1.8 *BEAM_FORMING_TI_OutArgs*

Description

This structure contains all the parameters which are output from the applet

Fields

Field	Data Type	Input/ Output	Description
iVisionOutArgs	IVISION_OutArgs	Output	Common outArgs for all ivison based modules

4.3 Peak Detection

The purpose of this applet is to identify the peaks in range and Doppler dimension, which provides the information about the distance and velocity of the object. This applet expects a buffer which is computed by doing range and Doppler FFT on the input radar cube data. The format of the input data is described later in the next section using PEAK_DETECTION_TI_BUFDESC_IN_ANTENNA_DATA_BUF enum. Apart from peak detection, this applet also supports configurable (enable/disable) TX decoding of the input data. TX decoding is used to separate individual transmitter data from the linear combination of multiple transmitter data. The output of this applet is described using PEAK_DETECTION_TI_OutBufOrder.
This applet computes the energy at each sample of input data cube and finds the peak using supported detection methods (Kindly refer PEAK_DETECTION_TI_METHOD enum for list of supported methods).

4.3.1 Data Structures

4.3.1.1 PEAK_DETECTION_TI_ErrorType

Description

Error codes returned by the this applets.

Fields

Enumeration	Description
PEAK_DETECTION_TI_ERRORTYPE_UNSUPPORTED_ CONFIGURATION	Indicates an un-supported configuration
PEAK_DETECTION_TI_ERRORTYPE_UNSUPPORTED_ BUF_DESCRIPTOR	Indicates an un-supported buffer descriptor
PEAK_DETECTION_TI_ERRORTYPE_UNSUPPORTED_ NOISE_LENGTH	Indicates an un-supported noise length
PEAK_DETECTION_TI_ERRORTYPE_UNSUPPORTED_ DOPPLER_DIMENSION	Indicates an un-supported Doppler dimension

4.3.1.2 PEAK_DETECTION_TI_InBufOrder

Description

User provides most of the information through buffer descriptor during process call. Below enums define the purpose of input buffer. For optimal performance all the buffers should be aligned to 128 byte boundary.

Fields

Enumeration	Description

Enumeration	Description
PEAK_DETECTION_TI_BUFDESC_IN_ANTENNA_DATA_BUF	This buffer descriptor provides the pointer to a buffer which contains the radar cube data (antenna data) after computing range and Doppler FFT on them. The properties of this buffer is described using PEAK_DETECTION_TI_BufferDescriptor structure. Each input sample contains real and imaginary data stored in interleave manner with each entry being 16 signed number. Please refer the description of this enum in ipeak_detection_ti.h to understand the arrangement of the data. It is important to note that only following parameters of this structure are used by this applet : bufDesc[]->bufPlanes[].buf bufDesc[]->bufPlanes[].accessMask
PEAK_DETECTION_TI_BUFDESC_IN_TOTAL	Total number of input buffers

4.3.1.3 *PEAK_DETECTION_TI_OutBufOrder*

Description

User provides most of the information through buffer descriptor during process call. Below enums define the purpose of output buffers. For optimal performance all the buffers should be aligned to 128 byte boundary.

Fields

Enumeration	Description
PEAK_DETECTION_TI_BUFDESC_OUT_BUFFER	This buffer will return different output based on the PEAK_DETECTION_TI_CfarCaDbCreateParams.detectionMethod. If detectionMethod == PEAK_DETECTION_TI_METHOD_CFARCA_DB then, This buffer stores the list of range and doppler coordinates for detected objects in interleaved manner with range in the upper 16 bit and doppler in lower 16 bit. Size of this buffer should be the worst case size of the coordinates which is range x doppler x 4. It is important to note that only following entries from this structure are used by this applet : bufDesc[]->bufPlanes[].buf bufDesc[]->bufPlanes[].accessMask If detectionMethod == PEAK_DETECTION_TI_METHOD_ENERGY_OUT then, This buffer will store the energy which is actually the sum of energy at each point for all the antennas in log2 format. Each entry is 16bit wide. size of this buffers should be range x doppler x 2. It is important to note that only following enteries from this structure are used by this applet : bufDesc[]->bufPlanes[].buf bufDesc[]->bufPlanes[].accessMask

Enumeration	Description
PEAK_DETECTION_TI_BUFDESC_OUT_ENERGY_BUFFER	This buffer stores the list of 16 bit container. It is important to note that only following entries from this structure are used by this applet : bufDesc[]->bufPlanes[].buf bufDesc[]->bufPlanes[].accessMask

PEAK_DETECTION_TI_BUFDESC_OUT_ANTENNA_DATA	This buffer is to store the antenna data corresponding to each detection. This is an optional buffer and is required only if user wants to get the antenna data corresponding to the detected object. The data in this buffer is arranged as numAntennas x numDetections. Buffer is expected to look as shown below :
	`   __|A0| A1| A2|........ |Ak |`
	`  D0|`
	`  D1|`
	`  D2|`
	`    |`
	`    |`
	`  Dn |`
	
	Here each entry is stored as 16-bit real and 16-bit imaginary part interleaved together. It is important to note that only following parameters of this structure are used by this applet :
	bufDesc[]->bufPlanes[].buf
	bufDesc[]->bufPlanes[].accessMask

PEAK_DETECTION_TI_BUFDESC_OUT_TOTAL	Total number of output buffers

4.3.1.4 *PEAK_DETECTION_TI_METHOD*

Description

Supported detection methods.

Fields

Enumeration	Description
PEAK_DETECTION_TI_METHOD_CFARCA_DB	Peak detection method use is Constant false alarm rate, cell averaging in log domain. In this method threshold level is calculated by estimating the level of noise floor around the cell under test (CUT). The cell under test (CUT) is compared against an average noise floor. The noise cells (which are used to compute the noise floor) are located on either side of the CUT. The CUT is separated from the noise cells by guard cells. There are noiseLen noise samples on either side of the CUT. Similarly there are guardLen samples on either side of the CUT, separating it from the noise samples. CFAR CA detector equation is CUT > Threshold * NoiseFloor In log domain this becomes log(CUT) - Log(NoiseFloor) > log(Threshold)

Enumeration	Description
PEAK_DETECTION_TI_METHOD_ENERGY_OUT	In this method energy sum across all the antennas is computed and returned via output bufdescriptor with enumeration number PEAK_DETECTION_TI_BUFDESC_OUT_BUFFER.

4.3.1.5 *PEAK_DETECTION_TI_CFARCA_DIRECTION*

Description

Supported directions for CFAR CA algorithm. Below enums define for this.

Fields

Enumeration	Description
PEAK_DETECTION_TI_CFARCA_DIRECTION_RANGE	CFAR CA will be applied along the range dimension

4.3.1.6 *PEAK_DETECTION_TI_BufferDescriptor*

Description

This structure is used to describe the buffers (input/output) to this applet. This descriptor represents the buffer in the unit of Antenna. We assume the same format across the antenna. Kindly refer section Appendix D: Radar Processing Data Flow to see some of the examples to understand the usage of this structure

Fields

Field	Data Type	Input/ Output	Description
numChunks	`uint8_t`	Input	If data for a single antenna cannot be represented within a maximum supported pitch then we will have to split the buffer into smaller chunk such that each chunk can represent the data of single antenna within with a pitch. This field represents this number of such chunks.
numHorzPoints	uint16_t	Input	Number of contiguous points in Horizontal direction corresponding to a single antenna. This is an array of size equal to numChunks field in this structure.
numVertPoints	uint16_t	Input	Number of contiguous points in vertical direction corresponding to a single antenna. The contiguity is defined in terms of the pitch field in this structure.
offsetBwAntennas	uint32_t	Input	Offset in terms of number of bytes to jump to next antenna data from the current base pointer. This field is also an array having numChunks entries. This should be word aligned
pitch	uint32_t	Input	Offset in terms of number of bytes to jump to the next sample corresponding to the same antenna. This field is also an array having numChunks entries. Pitch should be word aligned

Field	Data Type	Input/ Output	Description
numAntennas	uint16_t	Input	This field is meant for radar processing use case and should be set to one for normal FFT calculation. This tells the total number of antenna data which is coming with the buffer.

4.3.1.7 *PEAK_DETECTION_TI_CfarCaDbCreateParams*

Description

This structure defines the create time parameter needed for CFAR CA DB detection algorithm.

Fields

Field	Data Type	Input/ Output	Description
maxNoiseLen	uint16_t	Input	Maximum noise length. Only required if detectionMethod is PEAK_DETECTION_TI_METHOD_CFARCA_DB
maxGaurdLen	uint16_t	Input	Maximum gaurd length. Only required if detectionMethod is PEAK_DETECTION_TI_METHOD_CFARCA_DB

4.3.1.8 *PEAK_DETECTION_TI_CfarCaDbParams*

Description

This structure defines the parameter needed for CFAR CA DB detection algorithm.

Fields

Field	Data Type	Input/ Output	Description
noiseLen	uint16_t	Input	One sided noise Length. Noise Length should be power of 2.
gaurdLen	uint16_t	Input	One sided guard Len. These are the samples which are skipped from each side of Cell Under Test (CUT) for calculating the noise floor
constant1	uint16_t	Input	Two parameters C1 and C2 used for thresholding in CFAR CA detector. The equation is CUT > AverageNoiseFloor * C1/2^C2.
constant2	uint16_t	Input	Two parameters C1 and C2 used for thresholding in CFAR CA detector. The equation is CUT > AverageNoiseFloor * C1/2^C2

4.3.1.9 *PEAK_DETECTION_TI_AlgoCreateParams*

Description

This structure defines the detection algorithm specific create parameters of all the supported detection methods by this applet.

Fields

Field	Data Type	Input/ Output	Description
cfarCaDb	PEAK_DETECTION_TI_CfarCaDbCreateParams	Input	Create time parameter to be used if detection method is PEAK_DETECTION_TI_METHOD_CFARCA_DB

4.3.1.10 PEAK_DETECTION_TI_AlgoParams

Description

This union define the detection algorithm specific run time parameters of all the supported detection methods by this applet.

Fields

Field	Data Type	Input/ Output	Description
cfarCaDb	PEAK_DETECTION_TI_CfarCaDbParams	Input	Run time parameter to be used if detection method is PEAK_DETECTION_TI_METHOD_CFARCA_DB

4.3.1.11 PEAK_DETECTION_TI_CreateParams

Description

This structure defines the detection algorithm specific run time parameters of all the supported detection methods by this applet.

Fields

Field	Data Type	Input/ Output	Description
cfarCaDb	PEAK_DETECTION_TI_CfarCaDbParams	Input	Run time parameter to be used if detection method is PEAK_DETECTION_TI_METHOD_CFARCA_DB

4.3.1.12 PEAK_DETECTION_TI_CreateParams

Description

This structure contains all the parameters which this applet uses at create time

Fields

Field	Data Type	Input/ Output	Description
visionParams	`IVISION_Params`	Input	Common parameters for all ivison based modules
maxNumAntenna	uint16_t	Input	Maximum number of antennas in the system.
maxNumTx	uint16_t	Input	Maximum number of transmitter antennas in system

Field	Data Type	Input/ Output	Description
maxRangeDimension	uint16_t	Input	Maximum range dimension
maxDopplerDimension	uint16_t	Input	Maximum Doppler dimension
detectionMethod	uint8_t	Input	Method to be used for detection of objects. Refer PEAK_DETECTION_TI_METHOD to know the supported methods
enableAntennaDataOut	uint8_t	Input	If enabled the applet will extract the antenna data corresponding to the detection and returns as one of the output buffer (PEAK_DETECTION_TI_BUFDESC_OUT_ ANTENNA_DATA)
algoCreateParams	PEAK_DETE CTION_TI_Alg oCreateParam s	Input	Algorithm specific create time parameter. For supported algorithms refer PEAK_DETECTION_TI_METHOD

4.3.1.13 PEAK_DETECTION_TI_InArgs

Description

This structure contains all the parameters which controls the applet at run time

Fields

Field	Data Type	Input/ Output	Description
iVisionInArgs	`IVISION_In Args`	Input	Common inArgs for all ivison based modules
enableTxDecoding	uint8_t	Input	Enable/Disable Tx decoding. A value of 1 enables tx decoding
numTx	uint16_t	Input	Number of transmitters in the system
numRx	uint16_t	Input	Number of receivers in the system
offsetBwTx	uint16_t	Input	Offset in bytes between two transmitters
offsetBwRx	uint16_t	Input	Offset in bytes between two receivers
txDecodingCoefficients	int8_t	Input	Coefficients required for tx decoding. This is only required if enableTxDecoding = 1. This is a matrix ofnumTx X numTx elements. For Tx decoding data received at each transmitter is multiplied by Tx decoding matrix to decode individual
bufDescription	PEAK_DETE CTION_TI_Bu fferDescriptor	Input	The input buffers to this applet comes as part of input buf descriptors, but the arrange of data in these buffer is described using PEAK_DETECTION_TI_BufferDescriptor structure. This field hold the structure which describes the input buffers

Field	Data Type	Input/ Output	Description
algoParams	PEAK_DETECTION_TI_AlgoParams	Input	This structure is to provide Algorithm specific run time parameters. For supported algorithms refer PEAK_DETECTION_TI_METHOD

4.3.1.14 *PEAK_DETECTION_TI_OutArgs*

Description

This structure contains all the parameters which are given as an output by this applet at frame leve

Fields

Field	Data Type	Input/ Output	Description
iVisionOutArgs	`IVISION_OutArgs`	Output	Common outArgs for all ivison based modules
numDetections	`uint16_t`	Output	Number of detected objects

Appendix A: Compatibility Information

This section mentions the compatibility table with respect to previous release

Compatibility Parameters Settings (To achieve 1.18 compatibility)				
Applet/algo name	Parameter type	Parameter name		value to be set in 1.19
		1.18	1.19	
fft	uint8_t	NA	inputContainerFormat (FFT_TI_CreateParams)	FFT_TI_CONTAINER_FORMAT_16BIT
fft	uint8_t	NA	outputContainerFormat (FFT_TI_CreateParams)	FFT_TI_CONTAINER_FORMAT_16BIT
fft	uint8_t	NA	enable32bitsIntermResults (FFT_TI_InArgs)	0

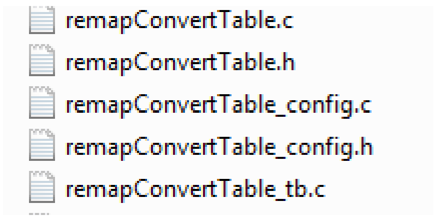
Appendix B: Remap Convert Table

Introduction

This document describes the details about the utility remapConvertTable, used to convert any (X,Y) table of a geometric transformation into the table format required by the remap applet.

Directory Structure Overview

The source files of the utility are located into the test folder of the remap_merge applet in *apps/remap_merge/test/src*. The files that used to build the utility are:


remapConvertTable.c
remapConvertTable.h
remapConvertTable_config.c
remapConvertTable_config.h
remapConvertTable_tb.c

With the main() entry point located in *remapConvertTable_tb.c* .

Building process

The makefile used to build the utility is located one directory above in *apps/remap_merge/test and* is named *makefileConvertTable.* This makefile is referred by a master makefile located in directory *apps* so when the user build the entire package from the root directory, the utility gets built as well. Hence, to build the executable, go to the root directory of the EVE SW release and type one of the following commands between quotes:

- ‘`gmake`’: build the utility to run on the target in release mode. The resulting executable file named *remapConvertTable.eve.out* is produced in directory *apps/remap_merge/test/elf_out .*
- ‘`gmake TARGET_BUILD=debug`’: build the utility to run on the target in debug mode. The resulting executable file named *remapConvertTable.eve.out* is produced in directory *apps/remap_merge/test/elf_out .*
- ‘`gmake TARGET_PLATFORM=PC`’: build the utility to run on the PC in release mode. The resulting executable file named *remapConvertTable.eve.out.exe* is produced in directory *apps/remap_merge/test/elf_out .*
- ‘`gmake TARGET_PLATFORM=PC TARGET_BUILD=debug`’: build the utility to run on the PC in debug mode. The resulting executable file named *remapConvertTable.eve.out.exe* is produced in directory *apps/remap_merge/test/elf_out .*

It is recommended to build the utility for PC since this table conversion is a step generally performed offline.

Usage

The utility takes various input arguments that are defined in a configuration file, instead of being passed through the command line. It is possible to have several configuration files, each one containing up to 10 sets of use cases. A master configutation list contained in the file *remapConvertTable_config_list.txt*, located in directory *apps\remap_merge\test\testvecs\config*, lists all the configuration files to be executed. By default it contains one configuration file ‘*remapConvertTableFishEye.cfg’*:

```
1 ../testvecs/config/remapConvertTableFishEye.cfg
```

0

Note that the last line must end with a ‘0’.

A configuration file has the following parameters:

- numTestCases: number of test cases listed in the configuration file. Up to 9 test cases can be specified in a configuration file. Each test case can be parametrized with the following parameters: `remapWidth#, remapHeight#, inputMapFile#, inputMapFileFormat#, qShift#, outputMapFile#, functionName#.` Where # corresponds to the index of the test case: 0, 1, 2, 3 … 9 .

- `remapWidth#`: width of the output image resulting from the geometric transform. For instance if the geometric transform rescales a 1080x720 image to 640x480 then remapWidth= 640.

- `remapHeight#`: height of the output image resulting from the geometric transform. For instance if the geometric transform rescales a 1080x720 image to 640x480 then remapHeight= 480.

- `blockWidth#`: processing block width.

- `blockHeight#`: processing block height.

- `inputMapFile#`: Relative path of the file containing the (X,Y) coordinates map defining the geometric transform. The path is relative to where the application is executed, e.g. the directory *apps\remap_merge\test\elf_out.* The file must contain remapWidth x remapHeight pairs of (X,Y) coordinates, listed in raster can format (lef to right, top to bottom), each one coded in 32-bits. The file format is specified by the next parameter inputMapFileFormat.

- `inputMapFileFormat#`: Can have value 0, 1 or 2.

  A value of 0 means that the inputMapFile is in binary format, with each X,Y coordinate occupying exactly 4 bytes (32 bits). So size of the file with this format would be 2 x 4 x remapWidth x remapHeight bytes.

  A value of 1 means that the inputMapFile is in text format, with each X,Y separated by one more several spaces as follow:

  X_0 Y_0

  X_1 Y_0

  X_2 Y_0

  …

  $X_{remapWidth-1}$ Y_0

  X_0 Y_1

  …

  $X_{remapWidth-1}$ Y_1

  …

  $X_{remapWidth-1}$ $Y_{remapHeight-1}$

  It is not possible to tell the size of the file in advance since each coordinate X,Y is in text format, with variable number of digits.

  A value of 2 means that the inputMapFIle is in text format, with each X,Y separated by a comma as follow:

  X_0, Y_0,

  X_1, Y_0,

  X_2, Y_0,

  …

  $X_{remapWidth-1}$, Y_0,

X_0, Y_1,

…

$X_{remapWidth-1}$, Y_1,

…

$X_{remapWidth-1}$, $Y_{remapHeight-1}$,

Note that spaces are allowed after each comma.

The reader is encouraged to have a look at the example input maps *inputMap_int.bin* (file format 0), *inputMap_int.dat* (file format 1) and *inputMap_int.txt* (file format 2) in directory *apps\remap_merge\test\testvecs\input.*

- `qShift#`: number of bit shift that was applied to the the (X,Y) coordinates in order to support fractional values. For instance with qShift=2, the coordinates (50.1, 100.7) must actually have been written in the file as (round(2^2 * 50.1), round(2^2 * 100.7))= (round(200.4), round(402.8))= (200, 403).
- `outputMapFile#`: Relative path of the output of the utility, which is the converted map. The path is relative to where the application is executed, e.g. the directory *apps\remap_merge\test\elf_out.* The file format is specified by the next parameter outputMapFileFormat.
- `outputFileFormat#`: Can have value 0 or 1.

  A value of 0 means the output is in binary format. The file format is explained in the next section.

  A value of 1 means the output is in *.c format, meaning that the file can be directly compiled as C source file with the other sources files of your application. The reader is encouraged to have a look at the example output map *outputMap_int1.c* in directory *apps\remap_merge\test\testvecs\output.*
- `functionName#`: This is only applicable if `outputFileFormat`=1, meaning output file is a *.c file. The *.c file will contain a definition of a function callable by the application to setup a structure and a pointer so the remap applet can apply the geometric transformation specified in the *.c file. The name of the function is defined here by the user so that it is possible to include several geometric transformations defined by several *.c file into the build without encountering any multiple symbol linker errors.

Content of the example file *remapConvertTable.cfg* is provided here:

```
numTestCases     = 4

remapWidth0   = 256
remapHeight0  = 128
blockWidth0 = 128
blockHeight0 = 8

qShift0       = 2
functionName0 = "fisheye_getLeftBlockMap"
inputMapFileFormat0 = 0
inputMapFile0       =  " ../testvecs/input/fisheyeMap256x128.bin"
outputMapFileFormat0 = 0
outputMapFile0      =  " ../testvecs/output/fisheyeMap256x128_conv.bin"

remapWidth1   = 256
remapHeight1  = 128
blockWidth1 = 128
blockHeight1 = 8

qShift1       = 2
functionName1 = "fisheye_getLeftBlockMap"
inputMapFileFormat1 = 0
inputMapFile1       =  " ../testvecs/input/fisheyeMap256x128.bin"
outputMapFileFormat1 = 1
outputMapFile1      =  " ../testvecs/output/fisheyeMap256x128_conv1.c"

remapWidth2   = 256
remapHeight2  = 128
```

```
blockWidth2 = 128
blockHeight2 = 8

qShift2       = 2
functionName2 = "fisheye_getLeftBlockMap"
inputMapFileFormat2 = 1
inputMapFile2       =  " ../testvecs/input/fisheyeMap256x128.dat"
outputMapFileFormat2 = 1
outputMapFile2      =  " ../testvecs/output/fisheyeMap256x128_conv2.c"

remapWidth3   = 256
remapHeight3  = 128
blockWidth3 = 128
blockHeight3 = 8

qShift3       = 2
functionName3 = "fisheye_getLeftBlockMap"
inputMapFileFormat3 = 2
inputMapFile3       =  " ../testvecs/input/fisheyeMap256x128.txt"
outputMapFileFormat3 = 1
outputMapFile3      =  " ../testvecs/output/fisheyeMap256x128_conv3.c"
```

How to use a converted map with the remap applet.

At creation time, the remap applet needs the following structure defined in file *apps/remap_merge/algo/inc/iremap_merge_ti.h* to be initialized:

```
typedef struct
{
  IVISION_Params visionParams;
  RemapParms     remapParams;
  uint8_t        enableMerge;
  Format         dstFormat;

} REMAP_MERGE_TI_CreateParams;
```

The member `remapParams` is defined as follow in *kernels\vlib\vcop_remap\inc\remap_common.h*:

```
typedef struct {
    Interpolation interpolationLuma;
    Interpolation interpolationChroma;
    uint8_t       rightShift;
    int32_t       sat_high;
    int32_t       sat_high_set;
    int32_t       sat_low;
    int32_t       sat_low_set;
    sConvertMap   maps;
    uint32_t      reserved[3];
} RemapParms;
```

All the members in above structures can be initialized independently from the geometric transform except for the member `maps` which is defined as structure `sConvertMap`:

```
typedef struct
{
  const void *srcMap;
  Size        mapDim;
  Size        srcImageDim;
  uint8_t     isSrcMapFloat;
  Format      srcFormat;
  Size        outputBlockDim;
  Size        inputTileDim;
  uint8_t    *outputBlkInfo;
  sTileInfo  *tileInfo;
  uint16_t    maxNumPixelsinTile;
  uint16_t    maxNumEvenPixelsinTile;
  uint16_t    maxNumOddPixelsinTile;
  uint32_t    tileInfoSize;
  Size        maxInputBlockDim;
  uint32_t    maxInputBlockSize;
  void       *blockMap;
  uint8_t     qShift;
} sConvertMap;
```

This structure depends on the geometric transform and members outputBlockDim, maxInputBlockDim, maxInputBlockSize, maxnumPixelsInTile, qShift need to be initialized.

Fortunately, this structure can be automatically initialized using one of the following two ways:

1) The converted map was saved as a *.c file.

   In this case, the application just needs to call the function contained in the generated *.c file and whose name was specified in the config file.
   In the example config file of the previous section, the function name was `stereovision_getLeftBlockMap()`. To see the definition of the function, one can open the file *outputMap_int1.c* and scroll down to the bottom:

```
void stereovision_getLeftBlockMap(unsigned char **pSrcBlkMap, unsigned int *blockMap_LEN,
sConvertMap *mapStruct){

  *pSrcBlkMap= (unsigned char*)&srcBlkMap[0];
  *blockMap_LEN= 1136064;
  mapStruct->outputBlockDim.width= 128;
  mapStruct->outputBlockDim.height= 8;
  mapStruct->maxInputBlockDim.width= 167;
  mapStruct->maxInputBlockDim.height= 26;
  mapStruct->maxInputBlockSize= 3072;
  mapStruct->maxNumPixelsinTile= 0;
  mapStruct->qShift= 2;
  return;

}
```

   We can see that the third argument of the function call is a pointer to the `sConvertMap` structure, which is automatically filled out.
   The first argument of the function is a pointer to the location where the pointer to the converted map will be stored. The second argument is the pointer to the location where the size of the converted map will be stored to. Both are required when calling the `process()` function of remap().

   An example of how this function is called from a top application can be found in the Vision SDK plugin file *remapMergeLink_algPlugIn.c*:

   At create time, `stereovision_getRightBlockMap()` is called at line 291:

```
stereovision_getRightBlockMap(&(pRemapMergeObj->srcBlkMap[channelId]),
                &(pRemapMergeObj->blockMapLen[channelId]),
                &(pAlgCreateParams->remapParams.maps));
```

   The variable `pRemapMergeObj->srcBlkMap[channelId]` and `pRemapMergeObj->blockMapLen[channelId]` are latter used right before process call at lines 576 to 578:

```
pInBufs->bufDesc[1]->bufPlanes[0].buf = (UInt8 *)pRemapMergeObj->srcBlkMap[channelId];
pInBufs->bufDesc[1]->bufPlanes[0].width          = pRemapMergeObj->blockMapLen[channelId];
pInBufs->bufDesc[1]->bufPlanes[0].frameROI.width = pRemapMergeObj->blockMapLen[channelId];


…

status = AlgIvision_process(pRemapMergeObj->handle[channelId],
                                         pInBufs,
                                         pOutBufs,
                                         (IVISION_InArgs *)pInArgs,
                                         (IVISION_OutArgs *)pOutArgs
                                        );
```

2) The converted map was saved as a binary file.

The structure of the converted map binary file is as follow:

Byte Offset	Content
0	MAP_SIZE= Length of converted map in bytes
4	Converted map of size MAP_SIZE This section contains the structure sConvertMap as well as the array srcBlkmap

Total size of the file is 4 + MAP_SIZE bytes .

The following code shows how to read that binary file:

```
        if ((fid= fopen(“convertedMap.bin,"rb"))== NULL) {
                    PRINT_ERRORE_MSG();
                    goto EXIT;
                }

        fread((void*)&lengthOfConvertedMap, 4,  1, fid); /* first we read MAP_SIZE into lengthOfConvertedMap */

        if ((convertedMap = (uint8_t*)malloc(lengthOfConvertedMap)) == NULL){ /* we allocate memory to store the
converted map */
            status= IALG_EFAIL;
            goto EXIT;
        }

        fread((void*)convertedMap, 1,  lengthOfConvertedMap, fid); /* we read the converted map into the allocated
memory */
        fclose(fid);

        rectify_getBlockMap(&srcBlkMapRight, &blockMap_LEN, &rightParams->maps, convertedMapRight);
```

The function `rectify_getBlockMap()` is given below:

```
void rectify_getBlockMap(uint8_t **srcBlkMap, uint32_t *blockMap_LEN, sConvertMap *maps, uint8_t *convertedMap) {

    uint32_t i;
    uint8_t *p;
    p= (uint8_t*)maps;

    for (i=0; i< sizeof(sConvertMap);i++)
        *p++= *convertedMap++;

    p= (uint8_t*)blockMap_LEN;
    for (i=0; i< sizeof(*blockMap_LEN);i++)
        *p++= *convertedMap++;

    *srcBlkMap= convertedMap;
}
```

How to verify a converted map

To verify whether the converted map works well with remap, a test utility called `remapExecute` is provided in the same directory *apps\remap_merge\test\elf_out.*
Like the convert table utility, it parses any configuration file listed in *apps\remap_merge\test\testvecs\config\remapConvertTable_config_list.txt* .
The default content is:

```
1 ../testvecs/config/remapConvertTableFishEye.cfg
0
```

A configuration file has the following parameters:

- numTestCases: number of test cases listed in the configuration file. Up to 9 test cases can be specified in a configuration file. Each test case can be parametrized with the following parameters: `remapWidth#`, `remapHeight#`, `originalMapFile#`, `convertedBinMapFile#`, `inputFile#`, `outputFile#`. Where # corresponds to the index of the test case: 0, 1, 2, 3 … 9 .

- `remapWidth#`: width of the output image resulting from the geometric transform. For instance if the geometric transform rescales a 1080x720 image to 640x480 then remapWidth= 640.

- `remapHeight#`: height of the output image resulting from the geometric transform. For instance if the geometric transform rescales a 1080x720 image to 640x480 then remapHeight= 480.

- `originalMapFile#`: Relative path of the file containing the (X,Y) coordinates map defining the geometric transform. The file format must be binary. Basically it is the file that was passed to convertTable as input. Relative path of the output of the utility, which is the converted map. The path is relative to where the application is executed, e.g. the directory *apps\remap_merge\test\elf_out.* The file must contain remapWidth x remapHeight pairs of (X,Y) coordinates, listed in raster can format (lef to right, top to bottom), each one coded in 32-bits.

- `convertedBinMapFile#`: This is the output map file from convertTable, in binary format.

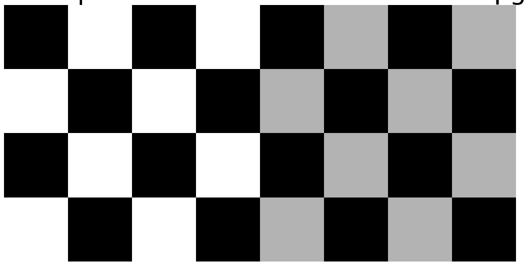
- `inputFile#`: this is the input pgm file, to which the geometric transform specified by `convertedBinMapFile#` will be applied.

- `outputFile#:` the resulting image from remap applied to `inputFile#`.

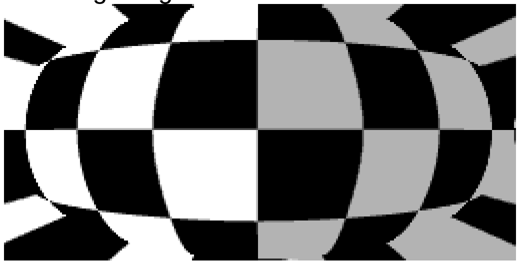
The sample configuration file provided in the release has the following content:

```
numTestCases     = 1

remapWidth0   = 256
remapHeight0  = 128
originalMapFile       =  " ../testvecs/input/fishEyeMap256x128.bin"
convertedBinMapFile      =  "output/fishEyeMap256x128_conv.bin"
inputFile = " ../testvecs/input/checkerboard256x128.pgm"
outputFile = " ../testvecs/output/fish_eye_checkerboard256x128.pgm"
```

The input file `checkerboard256x128.pgm` is a checker-board pattern:



After table conversion and remap execute, the output file `fish_eye_checkerboard256x128.pgm` should contain the following image:



When executing in target mode, on EVM, the console window will also display performance measurements. The expected printout for the fish eye lens example is:

```
TEST_REPORT_PROCESS_PROFILE_DATA : TSC cycles = 251782, SCTM VCOP BUSY cycles = 132096, SCTM VCOP
Overhead = 0
I/O Bytes :        0 (        0 +        0 ) :        0 (        0 +        0 )
Graph create: 140449 ARP32 cycles | Execution (control + VCOP + EDMA):   7.68 VCOP cycles |
Execution (VCOP only):   4.03 VCOP cycles
: TSC cycles = 251782, SCTM VCOP BUSY cycles = 132096, SCTM VCOP Overhead = 0
I/O Bytes :        0 (        0 +        0 ) :        0 (        0 +        0 )

Graph create: 101627 ARP32 cycles | Execution (control + VCOP + EDMA):   5.12 VCOP cycles/pixel |
Execution (VCOP only):   3.41 VCOP cycles/pixel
```

Please ignore the first line starting with `TEST_REPORT_PROCESS_PROFILE_DATA`.
What is important is the second line:

```
Graph create: 101627 ARP32 cycles | Execution (control + VCOP + EDMA):   5.12 VCOP cycles/pixel |
Execution (VCOP only):   3.41 VCOP cycles/pixel
```

Basically `Graph create` is the number of cycles to create the BAM graph corresponding to the remap algorithm. This is a one-time event, which does not happen every frame, but would be part of the system initialization.
The execution cycles are of two types: the total execution time, which includes control + VCOP + EDMA cycles and VCOP only execution time. If the frame is large, the total execution time should be pretty close to VCOP only execution time because the impact of control overhead is less. With the fish eye example, since the image is small, the overhead is pretty large: (5.12 – 3.41)/3.41= 50%.

If an error message such as:

```
Remap initialization error: decrease blockWidth and blockHeight values for the use case processed
by remapConvertTable
Error use case #0, stopped processing
```

is displayed then the parameters `blockWidth` and `blockHeight` in the convert table configuration file need to be reduced.

Matlab utility to generate an example LUT for fish-eye lens transform

To help generating example LUTs, the matlab utility *createFishEyeLUT.m* is provided in directory *apps\remap_merge\test\utils*. It allows to play around with creating LUTs that correspond to fish-eye geometric transformation. User is free to modify the utility to generate LUT for other types of geometric transformation.
Dimensions of the generated map is set by the variables NC, NR, NPIX and derived as follow: image width is 2* NC * NPIX and image height is 2 * NR * NPIX.
The utility will generate:

- The (X,Y) LUT map in binary format, 32-bits for X, 32-bits for Y, into *../testvecs/input/fishEyeMap<WIDTH>x<HEIGHT>.bin*
- The (X,Y) LUT map in text format, into *../testvecs/input/fishEyeMap<WIDTH>x<HEIGHT>.txt*
- The (X,Y) LUT map in dat format, into *../testvecs/input/fishEyeMap<WIDTH>x<HEIGHT>.dat*

For testing purpose, synthetic input images are also generated to be used with remapExecute application:

- A gray checker board pattern of size W=2* NC * NPIX x H=2 * NR * NPIX into *../testvecs/input/checkerboard_gray_<WIDTH>x<HEIGHT>.pgm*
- A gray checker board pattern of size W=2* NC * NPIX x H=2 * NR * NPIX into *../testvecs/input/checkerboard_gray<WIDTH>x<HEIGHT>.y*
- A color checker board pattern of size W=2* NC * NPIX x H=2 * NR * NPIX into *../testvecs/input/checkerboard_color<WIDTH>x<HEIGHT>.yuv*

Appendix C: Template Matching Normalization Function

Introduction

This appendix contains an example function that implements the normalization step that must be applied to the template image before passing it to the template matching applet.

This normalization step calculates the means of the template before subtracting it from every pixel values to produce the normalized template. In addition to that, the output is produced in Q format.

Also the pseudo-variance of the template is calculated alongside and returned at the exit of the function. This pseudo-variance corresponds to the constant K present in the normalized cross-correlation formula:

$$c(u,v) = \frac{\sum_{x,y} (f(x,y) - \bar{f}_{u,v}) \cdot t'(x-u, y-v)}{K \cdot \sqrt{\sum_{x,y} (f(x,y) - \bar{f}_{u,v})^2}}$$

Source code

```
/* Calculate the mean and subtract it from every pixel */
uint32_t normalizeTemplate(
        int16_t *dst, /* Destination pointer to the 16-bits template output in Q format */
        uint8_t *src, /* Source pointer to the 8-bits template image */
        uint32_t width, /* width of the template in number of pixels */
        uint32_t height, /* height of the template in number of rows */
        uint32_t stride, /* stride of the template in number of pixels */
        uint8_t sizeQshift, /* Number of fractional bits used to represent
1/(templateWidth*templateHeight) */
        uint8_t qShift) /* Number of fractional bits used to represent the output */
    {

    int32_t tAver, sum= 0;
    uint32_t tVar, w, h;
    int32_t t_p;
    uint16_t qValDiffHalf, qShiftDiff, qValHalf;

    qShiftDiff= sizeQshift - qShift;
    qValDiffHalf= 1<<(qShiftDiff -1 );
    qValHalf= 1<<(qShift - 1);

    memset(dst, 0, stride * height);

    /* Add all the pixel values together */
    for (h= 0; h < height; h++) {
        for (w= 0; w < width; w++) {
            sum += src[w + h*width];
        }
    }

    /* Caculate the mean by dividing the sum of all the pixels by the number of pixels in the
template, result has 'sizeQshift' fractional bits */
    tAver= ((sum<<sizeQshift) + (width*height)/2)/(width*height);
    tVar= 0;
    for (h= 0; h < height; h++) {
        for (w= 0; w < width; w++) {
            /* Subtract the mean from every pixel values. Convert to Q format by left shifting the
input template valye by qShift and
             * right shifting the mean by qShiftDiff
             */
            t_p= (src[w + h*width]<<qShift) - ((tAver + qValDiffHalf)>>qShiftDiff);
            dst[w + h*stride]= t_p;
            /* Calculate the pseudo variance of the template, which represents the constant value
K used in the normalized cross-correlation
             * Result is in Q format
             * */
            tVar+= ((unsigned int)t_p*t_p + qValHalf) >> qShift;
        }
    }

    return tVar; /* Return the pseudo-variance */
}
```

Q-format

The function uses the parameters qShift and sizeQshift in order to increase the precision of the results.

sizeQshift is used when dividing the sum of the pixels by the number of pixels to produce the mean value:

$$mean = \frac{\left(\sum_{x,y} t(x,y)\right) << sizeQshift}{templateWidth \cdot templateHeight}$$

A good value for sizeQshift is log2(templateWidth * templateHeight).

qShift is used for formatting the final output and the pseudo-variance into the corresponding Q format. If qShift= 8 then the normalized template is formatted in Q8.8 format and the pseudo-variance in Q24.8 format.

A good value for qShift is qShift= min(7, (22 - log2(templateWidth*templateHeight))/2);

The following constraints were applied in order to derive the formula:

$$c(u,v) = \frac{\sum_{x,y} \overbrace{\left(f(x,y) - \bar{f}_{u,v}\right)}^{\text{A: (9 + qShift) bits <= 32}} \cdot \overbrace{t'(x-u, y-v)}^{\text{B: (9 + qShift) bits <= 16 bits}}}{K \cdot \sqrt{\sum_{x,y}\left(f(x,y) - \bar{f}_{u,v}\right)^2}} \longrightarrow$$

C:
2 * (9 + qShift) + log2(templateWidth * templateHeight bits)   <=   40 bits

Constraint A means that at minimum, 9 bits are required to hold the result of the substraction between the 8-bits input and the mean value. Since the mean value is in Q-format, an additional qShift bits are required. To meet EVE implementation constraint, the total number of bits must fit within32 bits, that’s why we have (9 + qShift) bits < 32 .

Constraint B is similar to constraint A since t' represent the normalized template. The only difference is that the result of the normalized template is stored in 16-bits, not 32-bits.

Constraint C is related to the fact that the internal registers of EVE are 40-bits and thus the final expression

$$\sum_{x,y}(f(x,y)-\bar{f}_{u,v})\cdot t'(x-u,y-v)$$

must fit within 40 bits. Inside the expression, we have a summation of templateWidth * templateHeight elements, which increase the number of bits by log2(templateWidth * templateHeight), and each elements of the summation is a product of two values of bit depth (9 + qShift) bits.

Finally, we must respect these three constraints:

A: (9 + qShift) bits <= 32

B: (9 + qShift) bits <= 16

C: 2 * (9 + qShift) + log2(templateWidth * templateHeight bits) <= 40

which is equivalent to:

A: qShift <= 23

B: qShift <= 7

C: qShift <= (22 - log2(templateWidth*templateHeight))/2

Or qShift= min(23, 7, (22 - log2(templateWidth*templateHeight))/2)= min(7, (22 - log2(templateWidth*templateHeight))/2)

Appendix D: Radar Processing Data Flow

Introduction

This Appendix describes the details of how to use various radar Applications (applets) on EVE to implement the radar processing flow on EVE.

Details

Typical automotive radar systems need to identify objects with their world position and relative velocity to ego vehicle. Figure 2 shows a main processing blocks of automotive radar signal processing chain. First step is to find the distance of the object which can be found by doing an FFT in horizontal direction. Second step is to find the velocity of the object by computing the FFT in vertical direction. The energy peaks after computing range and Doppler FFT will give us distance and velocity of the object. Once we detect object next step is to find the angle of the object which can be found using beam forming technique. There can be additional pre-processing steps like interference zero out, dc-offset removal, windowing etc which can be applied before computing the FFT. Our implementation supports 1-6 processing blocks as shown in Figure 2

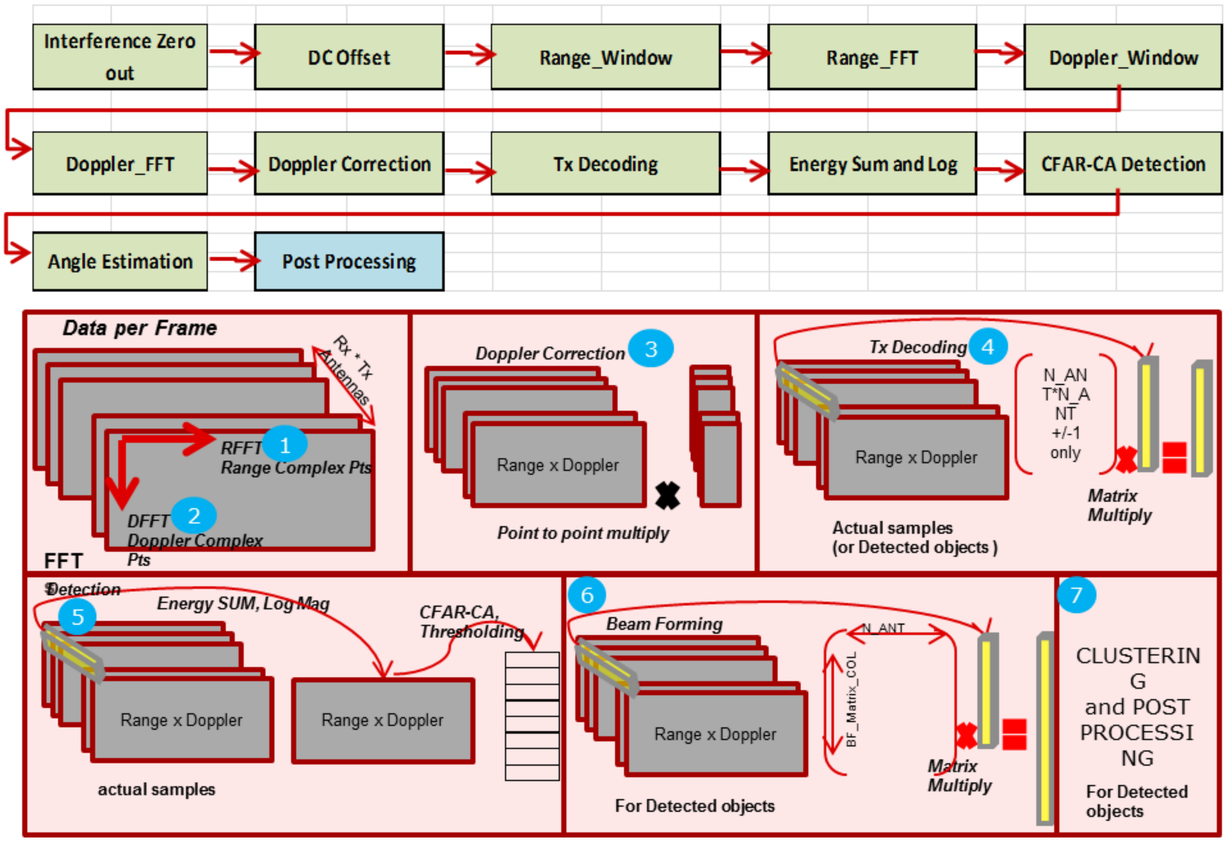


Figure 2 : Automotive Radar processing Blocks with data representation

To achieve this flow three applets are provided as part of EVE SW offering as listed below

- FFT (both range and Doppler)
- Peak Detection
- Beam Forming

User shall call the above applets one after the other such that the output of one applet goes as an input to the next one. Lets look at each of them one by one and see how they are connected.

1. Range and Doppler determination

The first step in radar processing is to find FFT of the input radar cube in horizontal direction, which will give the distance (range) of the object followed by the FFT in vertical direction, which will give the velocity (Doppler) of the object. This can be achieved by using FFT applet in two passes.

We will need to create two instances of FFT applet one configured in horizontal direction and the other configured in vertical direction. The same can be achieved by creating two instances of FFT applet with different create time parameters. User shall set the direction of the FFT using FFT_TI_CreateParams->fftdirection parameter during instance creation of the FFT applet.

Input to the FFT applet is provided using FFT_TI_BufferDescriptor. In this section we will take one example and describe what parameters needs to be provided for the buffer descriptor. For details of each parameter of using FFT_TI_BufferDescriptor, please refer 4.1.1.8

Figure 3 below shows layout of the output of ARxx sensors in the memory. Here r is range dimension and d is Doppler dimension. offsetBwAntenna indicates the offset from the one antenna to the next antenna. Here Rx_mTx_n represents data received by mth receivers of nth transmitter. For the cases when FFT is used as general purpose FFT instead of radar specific FFT, user can visualize the same data with number of antennas as 1.

For this input following would be the configuration of the FFT_TI_BufferDescriptor :

numChunks	= 1
numHorzPoints[0]	= r (Range dimension)
numVertPoints	= d (Doppler dimension)
offsetBwAntennas[0]	= offsetBwAntenna
numAntennas	= m x n (m transmitters and n receivers)
pitch[0]	= offsetBwAntenna * numAntennas
irregularityFrequency	= 0

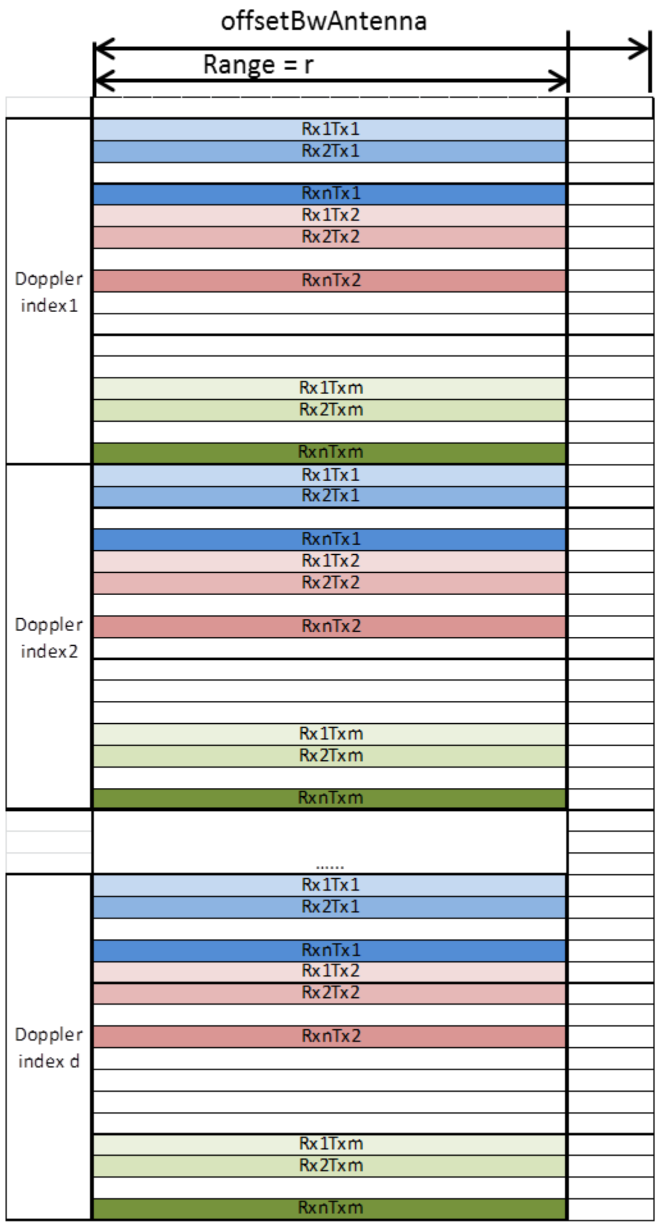


Figure 3 : Output of ARxx sensor

This buffer is given as an input to the first instance of the FFT applet. Apart from the input, there are other parameters like interference zero out, dc offset removal, windowing which user can enable /disable certain using inArgs of the applet. Please refer 4.1.1.10 various parameters, which can be configured at run time.

The output of the FFT for this instance shall provide us FFT in range dimension. This output shall be given as an input to the next instance of FFT which should be configured to be in vertical direction (Doppler dimension).

Figure 4 shows the output of range FFT. It is is also an input to Doppler FFT. This figure is just an example with three chunks. Number of chunks shown in Figure 4 is just an example. In general depending on the configuration number of chunks numChunks can be any number greater or equal to 1. As a user, one need not bother about the actual format of this buffer as this can be directly given as an input to the next instance of the FFT (Doppler FFT). Also buffer1, buffer2 and buffer3 can be placed anywhere in memory (even though they are shown side by side in the figure). The number of horizontal point (n1,n2,n3) in each of the buffers would be dependent on the configuration and would be determine by the algorithm itself. Typical values would be 8,16 32 etc.

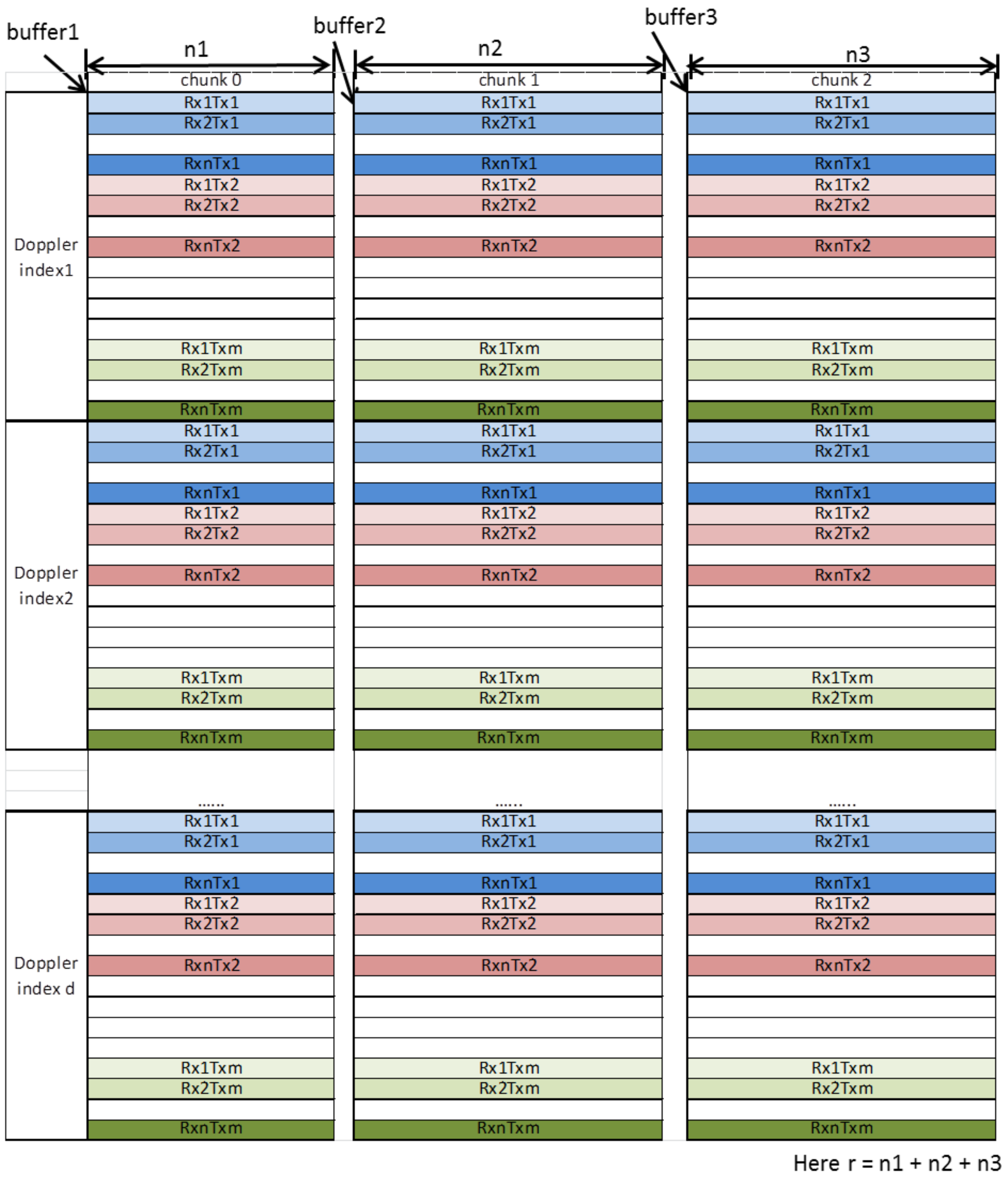


Figure 4 : Output of Range FFT/ Input to Doppler FFT

The buffer descriptor for the output of range FFT (input to Doppler FFT) is as follows :

numChunks = 3

numHorzPoints[] = {n1,n2,n3}

numVertPoints = d

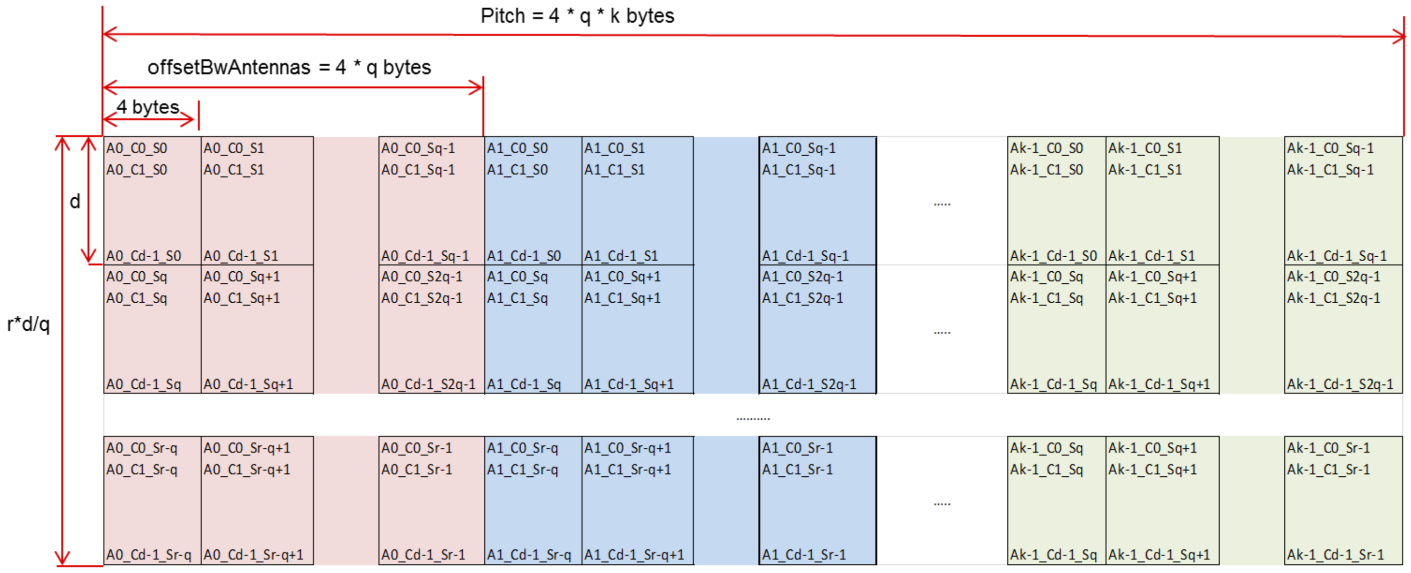
offsetBwAntennas[] = {n1 * 4,n2 * 4,n3 * 4} *

numAntennas = m x n

pitch[] = { n1 * 4* numAntennas, n2 * 4* numAntennas , n3 * 4* numAntennas } *

* multiplication by 4 in above example is because each element is a complex number with each real and imaginary of 2 bytes.

This buffer shall be given as an input to the Doppler FFT (Vertical direction FFT). The output of this instance is arranged in such a manner so that it will help in processing other blocks in radar processing chain. The output of this instance shall be as showin in Figure 5.



Here,
k = Number of antennas
r = range dimension
d = doppler dimension
q = 8,16 or 32
Ak_Ca_Sb = Antenna k , Doppler Index a , sample b data
Multiplication by 4 in above example is because each element is a complex number with each real and imaginary of 2 bytes.

Figure 5 : Output of Doppler FFT

Following is the buffer descriptor of the output of Doppler FFT:

numChunks = 1

numHorzPoints[0] = q

numVertPoints = (r / q) * d

offsetBwAntennas[0] = q * 4

pitch[0] = q * 4 * numAntennas

numAntennas = k (= m xn)

After the above two passes user shall get the FFT of input data in both range and Doppler dimension.

2. Peak detection :

Objective of Peak detection applet is to find the peaks in 2D FFT which was calculated in previous step. Input to peak detection applet is the output of Doppler FFT as shown in Figure 5. Peak detection uses CFAR CA algorithm to detect peaks. The applet first find the peaks in range dimension, followed by finding peaks only for the rows where range peaks are found in Doppler dimension. This applet supports an optional tx-decoding step which can be used if the samples from ARxx are multiplexed before transmitting. The user shall provide the coefficients for tx decoding in Q15 format by accounting the division factor along with the coefficients.

There are total 3 output buffers of this applet out of which one is optional (only required if user wants to use beam forming applet). Buffer 0 and Buffer 1 are two output buffers which stores the list of the coordinates (Range, Doppler) at which peak is detected and the energy corresponding to it. These buffers are enumerated as PEAK_DETECTION_TI_BUFDESC_OUT_RANGE_DOPPLER_BUFFER and PEAK_DETECTION_TI_BUFDESC_OUT_ENERGY_BUFFER in the output buffer descriptior. This applet also provides a mode to extract the input data for all the detected coordinates, which can be used by the beam forming module. This buffer is shown in Figure 6 as buffer 2. In real application it is expected that this conversion happens outside EVE as the processing is not very friendly to EVE because of the random access patterns. To enable the data out in the same format as expected by beam forming applet, user shall set PEAK_DETECTION_TI_CreateParams.enableAntennaDataOut = 1. If enabled one of the output buffer will be as shown in Figure 6. The same is enumerated as PEAK_DETECTION_TI_BUFDESC_OUT_ANTENNA_DATA in the output buffer descriptor.

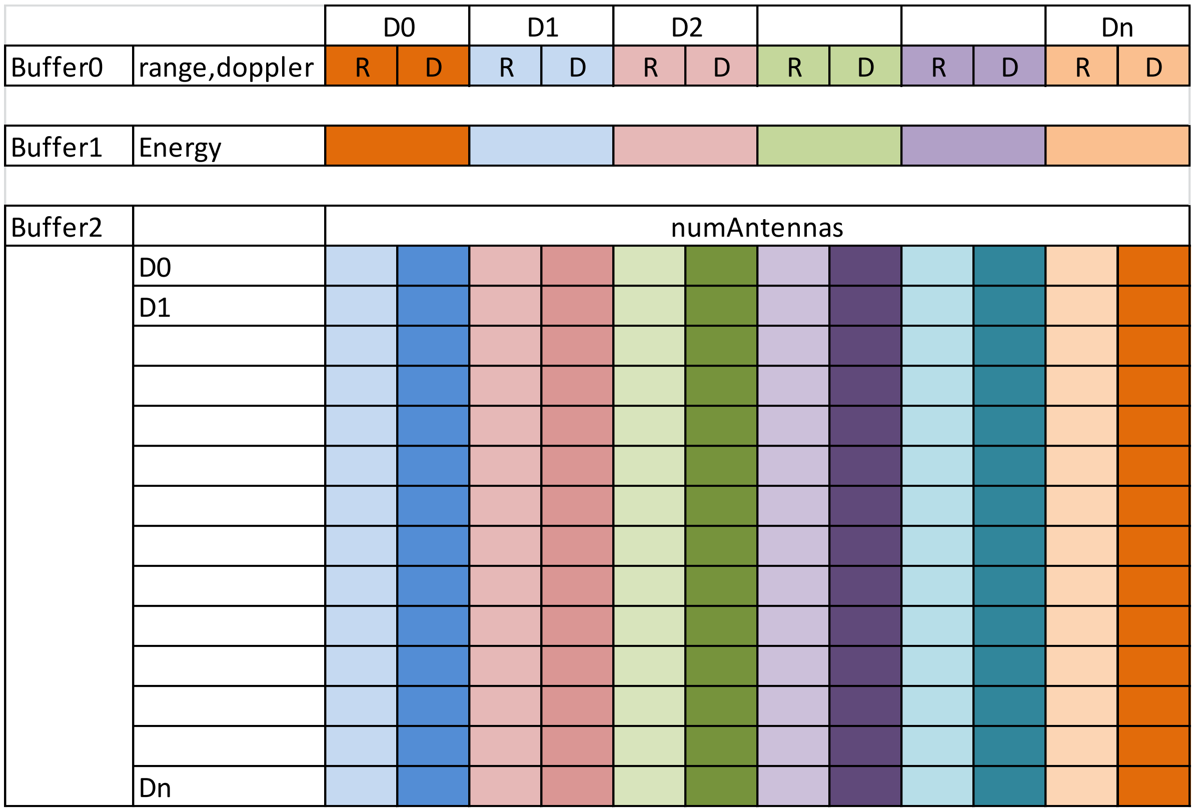


Figure 6 : Antenna Data out after Peak Detection

Here Dk denotes the kth detection. For each detection numAntenna samples corresponding to each antenna are extracted out from the original input data to this applet. This buffer is required for doing beam forming.

Peak Detection applet uses CFAR CA detector. In this algorithm threshold level is calculated by estimating the level of noise floor around the cell under test (CUT). The cell under test (CUT) is compared against an average noise floor. The noise cells (which are used to compute the noise floor) are located on either side of the CUT. The CUT is separated from the noise cells by guard cells. There are noise samples of length noiseLen on either side of the CUT.

Similarly there are guardLen samples on either side of the CUT, separating it from the noise samples. Figure 7 shows various parameters for CFAR CA algorithm.

CFAR CA detector equation is : CUT > Threshold * NoiseFloor.

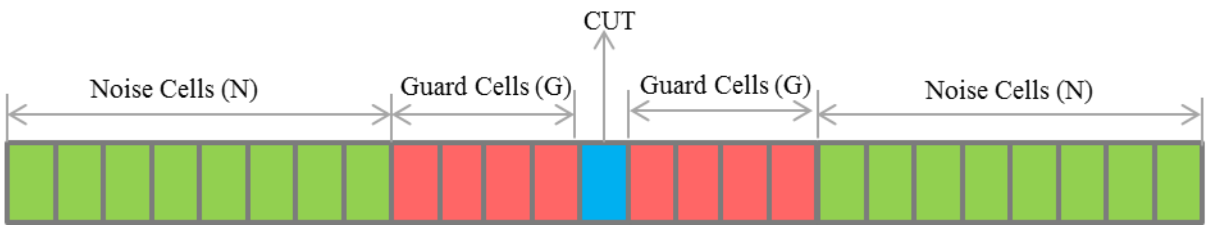


Figure 7 : CFAR CA Detection

Following are the input parameters (inArgs) for the input to peak detection applet as shown in Figure 5 assuming the tx and rx data is received as shown in Figure 3

PEAK_DETECTION_TI_InArgs.numTx = m

PEAK_DETECTION_TI_InArgs.numRx = n

PEAK_DETECTION_TI_InArgs.offsetBwTx = (4 * q * n) *

PEAK_DETECTION_TI_InArgs.offsetBwRx = (4 * q) *

PEAK_DETECTION_TI_InArgs.rangeDim = r

PEAK_DETECTION_TI_InArgs.dopplerDim = d

* multiplication by 4 in above example is because each element is a complex number with each real and imaginary of 2 bytes.

3. Beam Forming (Angle Estimation) :

The objective of this step is to find the angle of the detected objects in the previous step. Input to this applet is output of the peak detection applet which includes Range,Doppler coordinates and corresponding antenna data for those coordinates as shown in Figure 7 .
Beam forming applet supports two different kind of input buffers formats for range,Doppler coordinates. These are enumatred as BEAM_FORMING_TI_COORDINATE_BUF_FORMAT_1 and BEAM_FORMING_TI_COORDINATE_BUF_FORMAT_2. Format1 expects the input as a list of 4 elements range, Doppler, angle and energy in a single buffer, whereas Format2 expects input in 2 different buffers one containing list of range and Doppler coordinates and other containing the energy corresponding to them. If user wants to use the output of peak detection applet as described in the previous section then user shall set
BEAM_FORMING_TI_CreateParams.coordinateBufFormat= BEAM_FORMING_TI_COORDINATE_BUF_FORMAT_2

Apart from the coordinate buffer the applet also expects an antenna data buffer as showin in Figure 6. This buffer has dimension of numAntenna x numDetection and shall be provided as one of the input buffers to the beam forming applet. This buffer is enumerated as BEAM_FORMING_TI_BUFDESC_IN_ANTENNA_DATA_BUF in the input buffer descriptor.

Lets take an example in which user wants to detect angle in the range of -45 degrees to +45 dergrees at the resolution of 1 degree. In this case there will be total of 90 angles at which user wants to estimate the angle of the object. Each angle will corresponds to a steering vector of length equal to the number of antennas. So total dimension of steering matrix will be 90 * numAntennas.

Output of the beam forming applet is a list of structure which is described using BEAM_FORMING_TI_Coordinates as described in 4.2.1.3